VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

KRYPTOGRAFIE NA VÝKONOVĚ OMEZENÝCH ZAŘÍZENÍCH
CRYPTOGRAPHY ON CONSTRAINED DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Petr Šťovíček**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Dzurenda, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Petr Šťovíček ***ID:*** 214015
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Kryptografie na výkonově omezených zařízeních

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s možnostmi vývoje a implementace kryptografických protokolů na výpočetně omezených zařízeních, se zaměřením na IoT zařízení podporující OS RIOT (Raspberry Pi, Arduino apod.). Na tomto základě implementujete funkční systém sběru dat ze senzorů, který bude zajišťovat důvěrnost, autentičnost a integritu přenášených dat a současně bude RIOT kompatibilní.

DOPORUČENÁ LITERATURA:

[1] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] CVRČEK, Tadeáš. Kryptografie na platformě Arduino. Brno, 2020, 62 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce analyzuje možnosti aplikace kryptografických primitiv a protokolů na různá výpočetně a paměťově omezená zařízení. Následně implementuje systém bezpečného sběru dat ze senzorů. Práce ve své teoretické části rozebírá jednotlivé kryptografické algoritmy, operační systém RIOT a dostupné metody bezdrátového přenosu dat. Následně prezentuje výsledky výkonnostních testů různých kryptografických operací. Na tomto základě navrhne a realizuje systém, který zajišťuje důvěrnost, autentičnost a integritu přenášených dat.

KLÍČOVÁ SLOVA

Kryptografie, IoT, RIOT OS, WiFi, MQTT, ESP8266, ESP32, Raspberry Pi, Sběr dat, Bezpečnost

ABSTRACT

The bachelor thesis analyzes the possibilities of applying cryptographic primitives and protocols to various computationally and memory constrained devices. It then implements a secure data collection system from sensors. In its theoretical part, the work examines individual cryptographic algorithms, the RIOT operating system and available methods of wireless data transmission. It then presents the results of performance tests of various cryptographic operations. On this basis, it designs and implements a system that ensures the confidentiality, authenticity and integrity of transmitted data.

KEYWORDS

Cryptography, IoT, RIOT OS, WiFi, MQTT, ESP8266, ESP32, Raspberry Pi, Data Collection, Security

ŠŤOVÍČEK, Petr. *Kryptografie na výkonově omezených zařízeních*. Brno, 2020, 72 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Kryptografie na výkonově omezených zařízeních“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petrovi Dzurendovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Současný rozmach *Internet of Things* (IoT) systémů a průmyslu 4.0 je podmíněn připojením velkého množství omezených zařízení do internetu a dalších sítí. Tato zařízení nedisponují velkým výpočetním výkonem ani paměťovými prostředky. Zabezpečení těchto systémů bývá nízké či žádné. Cíl této práce je analyzovat současné možnosti IoT zařízení z pohledu kryptografické podpory a jejich možností zapojení do bezpečnostních systémů se zaměřením na bezpečný sběr dat ze senzorů. Na tomto základě navrhnout a implementovat funkční systém umožňující sběr dat ze senzorů, zajišťující důvěrnost, integritu a autentičnost přenášených dat.

Práce se v první kapitole věnuje základům kryptografie. Vymezuje základní pojmy, se kterými v dalších částech pracuje. Druhá kapitola se věnuje operačnímu systému RIOT, vysvětluje k čemu je vhodný, jak dosahuje své velikosti a popisuje jeho strukturu. Dále líčí podporované architektury a kryptografické knihovny. Také teoreticky rozebírá postup vývoje aplikací pro tento systém. Ve třetí kapitole analyzuje možné komunikační bezdrátové technologie. U každé metody přenosu se nachází její bezpečnostní zhodnocení. Čtvrtá kapitola je praktická, popisuje jakým způsobem zprovoznit vývojové prostředí, zkompilovat kód a nahrát ho na dané zařízení. V následující kapitole jsou benchmarkové testy. V úvodní části je popsána metodika měření. Následuje stručný popis jednotlivých desek. V dalších blocích jsou výsledky jednotlivých testů, včetně grafického znázornění. V každé sekci je popsáno jaká byla použita knihovna, případně problémy, které byly řešeny. Poslední část páté kapitoly vyhodnocuje naměřené hodnoty. Šestá, závěrečná kapitola, se věnuje systému bezpečného sběru dat ze senzorů. V úvodní části je popsán teoretický návrh systému, včetně jednotlivých technologií a entit v něm vystupujících. Následně je rozebrána konkrétní implementace. Na závěr závěrečné kapitoly byl proveden výkonnostní test celého systému.

1 Kryptografie na výkonově omezených zařízeních

Kryptografie (z řeckého *kryptós* - skrytý a *gráphein* - psát) je vědní obor zabývající se šiframi. Tedy převodem zpráv do podoby, jejíž přečtení je podmíněno speciální znalostí (např. znalost klíče, konstrukce šifry). Doplňkem kryptografie je kryptoanalýza, která hledá způsoby prolomení těchto systémů [1]. Tyto obory mezi sebou vedou neustálý boj, spočívající ve vymýšlení nových algoritmů a následně hledání jejich slabin. Většina šifer není neprolomitelná, je tedy možné hrubou silnou vypočítat utajovanou zprávu. Čas potřebný k tomuto výpočtu musí být natolik velký, aby byl v rozumném čase neproveditelný. Výkon počítačů však neustále exponenciálně roste [2] a je tedy nutné šifry tomuto růstu přizpůsobit. Bezpečnost lze navýšit například použitím delšího klíče nebo návrhem nové bezpečnější šifry. Tento proces sebou většinou nese i zvýšení nároků na výpočetní výkon zařízení, kde se šifrování provádí.

1.1 Výkonově omezená zařízení

Výkonově omezená zařízení nebo též omezená zařízení (z anglického constrained devices), jsou zařízení s nízkým výpočetním výkonem, malou pamětí, nízkou spotřebou a typicky nevelkými rozměry a cenou. Detailně jsou definovány v RFC 7228 [3]. Konkrétně se jedná o čipové karty, wearables (nositelná elektronika), *Radio-frequency identification* (RFID) tagy, senzory a mikrokontroléry. Další zařízení, která by se dala do této kategorie zařadit jsou různé chytré věci, tvořící chytré domácnosti. Mezi takové věci může patřit chytrá lednice, termostat, rolety, světla, garážová vrata, bezpečnostní systém, elektrické zásuvky, meteostanice, kamery, senzory teploty, vlhkosti a kvality ovzduší, atd. Tato zařízení dohromady v jedné síti tvoří IoT. Do konce roku 2020 bude do sítě internet připojeno 20,4 miliard IoT zařízení. V roce 2025 se jich očekává 75 miliard [4]. Často žádné nebo slabé zabezpečení z těchto zařízení dělají ideální cíle mnoha útoků. To může vést k řadě problémů. V lepším případě k tzv. zamrznutí přístroje či ztrátě soukromí majitele, v horším případě k tvorbě botnetů. V současné době existují IoT botnety tvořené desetitisíci kompromitovanými zařízeními, které na povel útočníka vytváří masivní *Distributed denial-of-service* (DDoS) útoky o velikosti několika set gigabitů za sekundu, kterým se dá jen obtížně bránit [5]. Tato zařízení hrají klíčovou roli i v průmyslu 4.0, kde by nedostatečné zabezpečení mohlo vést například k úniku informací či zastavení výroby. Zabezpečení je tedy pro další rozvoj IoT klíčové.

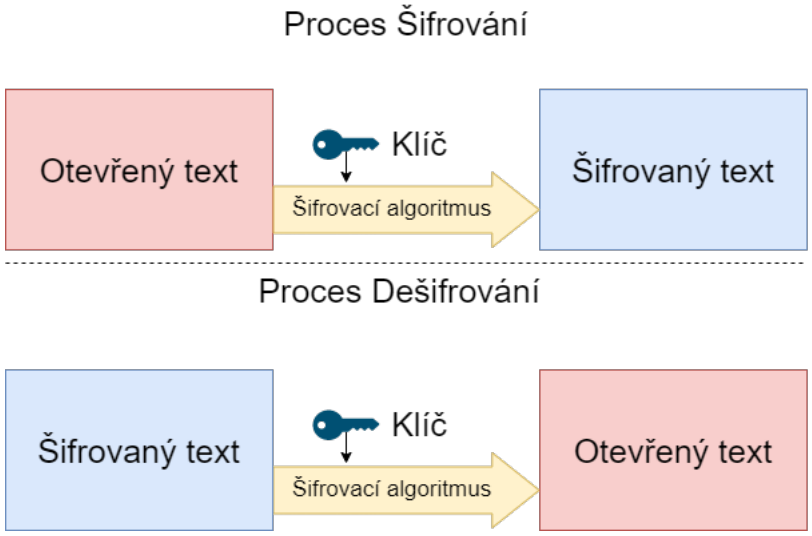
Aplikace kryptografických protokolů, používaných na klasických počítačích a telefonech, je na tato zařízení obtížná. Je to z důvodu nedostatečného výkonu a paměťových možností IoT zařízení, kdy by výpočet trval nepřijatelnou dobu. Proto je nutné použít algoritmy, které zajišťující dostatečnou bezpečnost, ale zároveň je možné je provádět i na takto omezených zařízeních.

1.2 Kryptografické algoritmy

Základní kryptografické nástroje se dají rozdělit do tří kategorií. Symetrické, asymetrické (využívající veřejný klíč) a bez klíče [1]. Pro větší přehlednost byly vybrány pouze tři podkategorie, kterým byly přiřazeny jednotlivé algoritmy.

1.2.1 Symetrické šifry

V symetrické kryptografii se používá stejný nebo lehce spočitatelný klíč pro šifrování i dešifrování. Její výhodou je rychlost, ta může být vyšší oproti asymetrické kryptografii i v několika set násobku. Nevýhodou je problematická distribuce klíče mezi stranami. Princip fungování symetrických šifer je možno vidět na obrázku 1.1.



Obr. 1.1: Fungování symetrických šifer

Data Encryption Standard (DES)

Algoritmus přijatý americkým standardizačním úřadem v roce 1976. Používá klíč o efektivní délce 56 bitů [6]. Několik desítek let sloužil jak ve státních organizacích, tak v soukromém sektoru. V roce 1999 se jej podařilo prolomit za 22 hodin 15

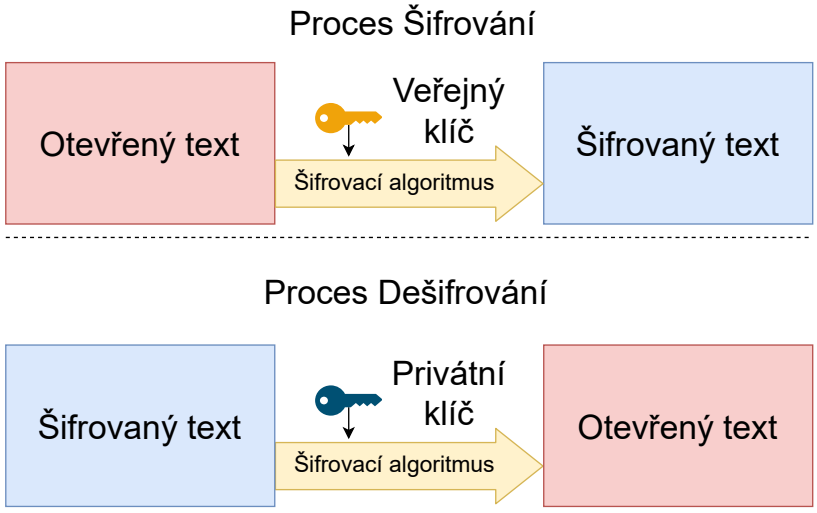
minut [7]. V reakci na nedostatečnou délku klíče v roce 1995 vznikl *Triple Data Encryption Algorithm, též TDES, TDEA, Triple DEA* (3DES), který klíč dokáže až ztrojnásobit na 168 bitů. Nicméně i ten je podle *National Institute of Standards and Technology* (NIST) považován za nebezpečný v systémech vzniklých po roce 2017 a měl by být vyměněn i ve všech ostatních aplikacích do roku 2023 [8].

Advanced Encryption Standard (AES)

Vítěz veřejné soutěže NIST z roku 1997 o nástupce šifry DES. Do té ji přihlásili autoři Joan Daemen a Vincent Rijmen pod názvem Rijndael [10]. V roce 2001 byl přijat jako standard pro šifrování. Uplatňuje klíč délky 128, 192 nebo 256 bitů, kterým šifruje bloky délky 128 bitů [11]. AES i v jeho nejslabší variantě je považován za bezpečný minimálně do roku 2030 [9].

1.2.2 Asymetrické šifry

Používají veřejný a soukromý (též privátní nebo tajný) klíč. Zprávu zašifrovanou veřejným klíčem lze dešifrovat klíčem soukromým a naopak. Na tomto principu je založeno množství různých protokolů. Proces šifrování a dešifrování je zobrazen na obrázku 1.2.



Obr. 1.2: Fungování asymetrických šifer

Rivest–Shamir–Adleman (RSA)

Jeden z prvních algoritmů aplikujících myšlenku asymetrické kryptografie. Byl patentován v roce 1983 [12]. Patent vypršel v roce 2000. Lze jej použít k šifrování i podepisování. Je založen na problému faktorizace velkých čísel. Při použití dostatečně

dlouhých klíčů (v současné době podle NIST alespoň 2048 bitů [9]) je považován za bezpečný. RSA se následně využívá v různých protokolech jako *Transport Layer Security* (TLS), *Pretty Good Privacy* (PGP) nebo při sestavování *Virtual private network* (VPN) tunelů [13].

Diffie–Hellman key exchange (DH)

Tento protokol byl publikován svými autory Whitfieldem Diffiem a Martinem Hellmanem v roce 1976 [14]. Umožňuje ustanovení tajného klíče mezi stranami po nezabezpečeném kanálu. Je postaven na problému diskrétního logaritmu. Při použití grupy velikosti alespoň 2048 bitů je považován za bezpečný [9].

Digital Signature Algorithm (DSA)

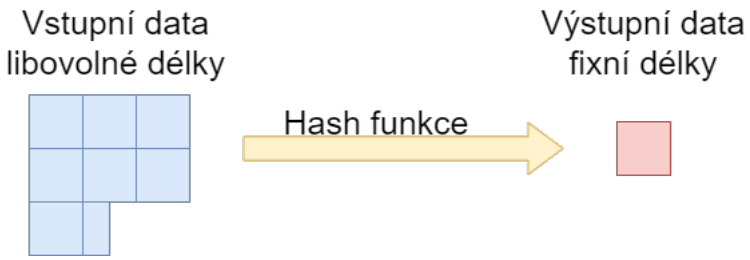
DSA je americký federální standard pro digitální podepisování. Byl navržen v roce 1991 institutem NIST. Opírá se o problém diskrétního logaritmu [15]. Nejde s ním šifrovat, pouze podepisovat. Využívá se například v OpenSSL, OpenSSH a GnuPG [16].

Kryptografie na bázi eliptických křivek

Eliptické křivky v kryptografii poskytují stejnou nebo vyšší úroveň zabezpečení za použití podstatně menšího klíče. To zkracuje výpočetní dobu, a tak i energetické nároky. Proto se hodí na zařízení s omezenými zdroji. Protokoly využívající eliptické křivky jsou například ***Elliptic-curve Diffie–Hellman*** **(ECDH)** nebo ***Elliptic Curve Digital Signature Algorithm*** **(ECDSA)**.

1.2.3 Hašovací funkce

Hašovací (též hash) funkce dokáží přetransformovat vstupní data libovolné délky na výstupní data (otisk, fingerprint) fixní délky. Jsou jednocestné, to znamená, že z výstupních dat nelze žádným způsobem odvodit data vstupní. Dále by měli být bezkolizní, to znamená, že různá vstupní data by neměla vést na stejný otisk. To je v praxi nedosažitelné, ale požaduje se, aby nalezení kolize bylo výpočetně velmi náročné, tedy prakticky neproveditelné. Hash funkce jsou mnohem rychlejší než symetrické šifry. Proto se dají použít třeba pro kontrolu celistvosti přenesených velkých souborů. Schéma fungování haší je vidět na obrázku 1.3.



Obr. 1.3: Fungování hašovací funkce

MD5

Představena Ronaldem Rivestem v roce 1992 jako nástupce MD4 [17]. Hašuje po blocích o velikosti 512 bitů, výstupní otisk má 128 bitů. Bylo nalezeno několik zranitelností, proto se pro většinu aplikací nedoporučuje [18]. Nadále se používá ke kontrole integrity stažených souborů, kdy si uživatel vypočítá haš staženého souboru a porovná ji s předpočítanou haší na serveru. Pokud se shodují, tak během přenosu pravděpodobně nedošlo k chybě. To ovšem zaručuje ochranu pouze před náhodnou chybou vzniklou při stahování, nikoliv jako ochrana před úmyslným zásahem do souboru. Útočník je totiž schopen soubor upravit tak, že uživateli vyjde haš stejná (vypočítat kolizi).

SHA2

Standard publikovaný Národní bezpečnostní agenturou v USA v roce 2001 [19]. Vytváří otisky délky 224 až 512 bitů. Všechny verze jsou dle NIST bezpečné [9]. Implementuje jej *Domain Name System Security Extensions* (DNSSEC) nebo systém datových schránek v České Republice [20].

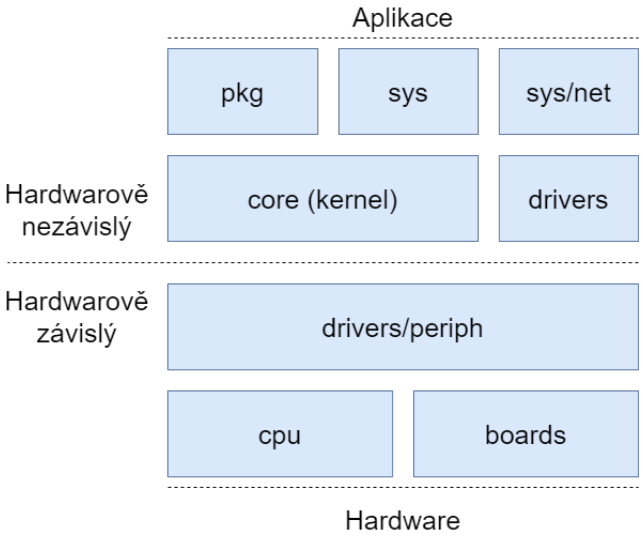
SHA3

Nejnovější hašovací funkce od institutu NIST z roku 2015 [21]. Podobně jako AES vyhrála veřejnou soutěž. Funguje odlišně než předchozí SHA2. Vnitřní konstrukce se popisuje jako „houba“. Stejně jako předchozí podporuje otisky různé délky od 224 až 512 bitů a je považována za bezpečnou [9].

2 RIOT OS

RIOT OS je open source operační systém s mikrojádrem, určený pro IoT zařízení. Díky své velikosti (1,5 kB RAM, 5 kB ROM) cílí na systémy, kde se klasický Linux nevejde. Těchto rozměrů dosahuje z důvodu optimalizace kódu a zejména tím, že se kompiluje vždy jen nezbytný kód k funkčnosti. Je to real-time, více vláknový modulární systém. Jeho hlavní výhodou je přenositelnost kódu, respektive nezávislost na hardwarové platformě. Stejný program lze spustit bez úprav na zařízeních různých výrobců [22].

Na obrázku 2.1 je vidět struktura RIOT. Každému bloku odpovídá složka, v ní se nacházejí jednotlivé moduly. Některé se připojí automaticky (jsou nutné k funkčnosti RIOT), většina ne. Tím dochází k úspoře místa na zařízení. Pomocí modulů se rozšiřuje funkčnost systému. Například, pokud je třeba použít některé kryptografické funkce, je nutné připojit modul crypto ve složce sys. Pokud je zapotřebí přidat podporu senzoru, provede se to připojením jeho modulu, tentokrát ze složky drivers. Hardwarové moduly (boards, cpu) obsahují kód specifický pro danou desku/procesor. Dle zvolené desky se hardwarové moduly většinou připojí automaticky a není je tedy nutné manuálně připojovat.



Obr. 2.1: Struktura RIOT OS

2.1 Podporované architektury

Většina populárních architektur a jejich zástupců jsou podporovány. Jejich kompletní výčet lze nalézt na GitHubu, ve složce cpu, respektive složce boards [23][24]. Mezi podporované architektury patří AVR, používaná mimo jiné deskami Arduino

Nano, dále ARM Cortex M3, jenž využívá Arduino Due nebo architektura ESP firmy Espressif.

Virtuální platforma RIOT native

Native je virtuální platforma/deska, umožňující emulaci RIOT OS. Při zvolení této desky během kompilace a nahrání kódu do zařízení se RIOT spustí jako proces na hostitelském počítači. Tato instance poté sdílí *Central Processing Unit* (CPU), *Random-Access Memory* (RAM), Flash a další zdroje s hostitelským strojem. Tím odstraňuje výkonové či paměťové limity klasických desek. Hodí se tak pro testování RIOT aplikací, bez nutnosti nahrání kódu na skutečný, kompatibilní hardware [25].

2.2 Vývoj aplikací

Vývoj aplikací běžících na RIOT OS probíhá pomocí *GNU's Not Unix!* (GNU) make. Vyvíjet lze z Linuxu, macOS nebo Windows pomocí virtualizačního nástroje. Vývojové prostředí poté tvoří minimálně kompilátor jazyka C a program make. Následně je možné stáhnout nejnovější verzi RIOT a naprogramovat první aplikaci. Pro nahrání kódu na zařízení je však nutný další program, který se může lišit v závislosti na desce, pro kterou je program určen. Celý proces by se dal rozdělit do následujících kroků:

Vývojového prostředí

To je možné zprovoznit několika způsoby. Nejspolehlivější je instalace všech potřebných programů a nástrojů přímo do počítače, ze kterého má vývoj probíhat. To však v současné době není možné na platformě Windows. Dalším možným způsobem je virtualizace, ta je možná i z operačního systému Windows. V tomto případě se všechny nezbytné programy nainstalují do linuxového systému, který je virtualizovaný z hostitelského (Windows) počítače pomocí programu typu VirtulBox. Samotný vývoj se poté nijak neliší od vývoje na skutečném stroji. Pouze při nahrávání kódu na zařízení je nutné přemostit sériové rozhraní ze skutečného do toho virtuálního. Detailnější popis je v kapitole 4.

Napsání kódu

Kód je možné psát v libovolném editoru, pro pohodlné programování je však ideální zvolit editor se zvýrazněním syntaxe jazyka C. Programuje se v jazyce C, ale je možné zapnout i podporu C++. Vytvoření aplikace pro RIOT je podmíněné

existencí minimálně dvou souborů. První je `main.c`, ve kterém se nachází kód aplikace. Jednoduchý program lze vidět na výpisu 2.1. Druhým povinným souborem je `Makefile`, obsahující instrukce pro překladač. Obsah takového souboru je vidět na výpisu 2.2. Ve druhém zmíněném souboru se pomocí proměnných nastavují parametry programu. Makefile musí minimálně obsahovat 4 položky, a to `APPLICATION` s názvem aplikace, `BOARD` určující zařízení, pro které se má kód přeložit, dále `RIOTBASE` obsahující absolutní cestu do adresáře RIOT a include souboru `Makefile.include` (viz výpis 2.2). Do tohoto souboru se také zapisují moduly a balíky, které se mají připojit do programu.

Výpis 2.1: Příklad souboru main.c

```
#include <stdio.h>                                  1
#include "xtimer.h"                                 2
                                                    3
int main(void)                                      4
{                                                   5
    while(1){                                       6
    puts("Hello␣World!");                           7
    xtimer_sleep(1);                                8
    }                                               9
    return 0;                                       10
}                                                   11
```

Výpis 2.2: Příklad souboru Makefile

```
# Název aplikace (povinné)                          1
APPLICATION = hello-world                           2
                                                    3
# Cílová vývojová deska (defaultně native) (povinné) 4
BOARD ?= native                                     5
                                                    6
# Absolutní cesta do RIOT adresáře (povinné)        7
RIOTBASE ?= /home/ubuntu/RIOT                       8
                                                    9
# Připojení modulů                                  10
USEMODULE += xtimer                                 11
                                                    12
# Připojení knihoven                                13
# USEPKG += micro-ecc                               14
                                                    15
# (povinné)                                         16
include $(RIOTBASE)/Makefile.include                17
```

Kompilace

Pro úspěšnou kompilaci je potřebné mít nainstalovaný správný tzv. toolchain. Ten se skládá z vývojových nástrojů a liší se v závislosti na platformě/architektuře, pro kterou se kód kompiluje. Toolchain obsahuje překladač (kompilátor) a případně další software nutný k správné funkci RIOT OS na desce daného výrobce. Některé platformy a příslušný překladač jsou vypsány v tabulce 2.1. Platforma native podporuje hned několik kompilátorů, proto není nutné použít ten uvedený v tabulce. Doporučený toolchain pro různé platformy se nachází na RIOT wiki v sekci Supported platforms [26].

Samotná kompilace se zahájí příkazem `make`, tím se kód zkompiluje pro desku, která je specifikována v souboru Makefile. Pokud má kompilace proběhnou pro jinou desku, je nutné přepsat proměnnou BOARD. To se provede buď v souboru Makefile nebo přímo při volání příkazu make následovně: `BOARD=arduino-nano make`. Označením desky se přiřadí správný toolchain. Názvy všech podporovaných desek lze najít na GitHubu ve složce boards [24].

Tab. 2.1: Některé platformy a odpovídající překladač

Platforma	kompilátor
native	GCC
ARM	GCC ARM Embedded
AVR	AVR GCC
ESP8266	Xtensa GCC

Nahrání kódu na zařízení (flash)

Nahrání kódu do většiny zařízení zajišťuje open source program `OpenOCD`. Některé zařízení vyžadují jiné nástroje. Ty jsou obvykle obsažené v rámci toolchainu. Například pro zapsání kódu do mikrokontroléru ATmega328p (Arduino nano) je potřeba doinstalovat nástroj `avrdude`. Také ESP8266 používá vlastní nástroj `esptool.py`, ten je však stejně jako OpenOCD obsažený v RIOT adresáři `dist/tools/`, a tak jej není třeba zvlášť instalovat.

Flash se provede příkazem `make flash`. Řídí se stejnou logikou jako kompilace výše. Je tedy nutné vybrat cílovou desku. Výběrem desky se automaticky přidělí správný nástroj pro nahrání kódu. Kromě desky je někdy třeba specifikovat port. Pomocí portu se určí sériové rozhraní, ke kterému je vývojová deska fyzicky připojena. Pokud se nechá prázdný, použije se defaultní hodnota, ta se liší v závislosti na použitém nástroji. Nástroj avrdude například použije `/dev/ttyUSB0`. Port se určí proměnnou PORT, je jí opět možné uvést do souboru Makefile nebo přímo do příkazu make. Celá kompilace i s nahráním kódu do zařízení by pak mohla vypadat následovně: `BOARD=arduino-nano PORT=/dev/ttyUSB1 make flash`.

Po nahrání kódu na zařízení je žádoucí navázat s ním komunikaci. Pro oboustrannou sériovou komunikaci RIOT používá aplikaci pyterm. Není třeba ji instalovat, nachází se v RIOT adresáři `dist/tools/`. Terminál se zavolá příkazem `make term`. Ke komunikaci je možné použít i jinou aplikaci.

Výše zmíněné kroky lze kombinovat, tj. lze vytvořit pouze terminál se nařízením nebo kód pouze zkompilovat (neodesílat na zařízení). Všechny lze také udělat současně `make flash term`. Po vyslání tohoto příkazu dojde ke kompilaci kódu, nahrání kódu do zařízení a vytvoření komunikačního kanálu.

2.3 Kryptografické knihovny

Knihovny se v RIOT OS připojují pomocí modulu Packages (pkg) [27]. K použití knihovny je nejdříve nutné ji zahrnout mezi ostatní moduly ke kompilaci. To se provede přidáním příkazu `USEPKG += <pkg_name>` do souboru Makefile. Následně je možné knihovnu naimportovat pomocí `#include "package.h"` přímo v kódu aplikace. Je možné použít i externí knihovny. To se provede přidáním jejich zdrojového kódu k ostatním knihovnám do složky `(RIOTBASE)/build/pkg/(PKG_NAME)` nebo přidáním knihovny přímo do složky daného projektu. Poté je možné s knihovnou pracovat stejným způsobem jako by byla přímo v RIOTu.

Následují některé podporované kryptografické knihovny.

micro-ecc

Knihovna vhodná k práci s body nad eliptickými křivkami. Je určena pro zařízení s omezeným výkonem. Implementuje ECDH a ECDSA algoritmy pro 8-bit, 32-bit a 64-bit architektury. Je napsána v jazyce C a je odolná vůči útokům postranními kanály. Pracuje s pěti standardními eliptickými křivkami secp160r1, secp192r1, secp224r1, secp256r1 a secp256k1. Zdrojový kód knihovny se nachází na GitHubu s licencí BSD 2-clause [28].

relic-toolkit

Relic je knihovna zaměřená na přenositelnost, flexibilitu a efektivitu. Implementuje řadu kryptografických protokolů (RSA, Rabin, ECDSA, Schnorrův podpis, Sakai-Ohgishi-Kasahara ID-based authenticated key agreement a několik dalších). Je dostupná na GitHubu, pod licencí Apache 2.0 and LGPL 2.1 [29].

tinycrypt

Knihovna orientovaná na omezená zařízení. Vyvíjena společností Intel. V necelých sedmi tisících řádcích kódu implementuje hašovací funkce jako SHA-256, blokovou šifru AES-128 v různých módech, výměnu klíčů ECDH nebo digitální podpis pomocí ECDSA. Implementace posledních dvou zmíněných standardů je založena na knihovně micro-ecc [30]. Zdrojový kód knihovny je i s podmínkami použití k dipozici na GitHubu [31].

3 Metody bezdrátového přenosu dat

Tato kapitola se zabývá analýzou bezdrátových technologií k bezpečnému přenosu dat mezi koncovým zařízením (senzorem) a serverem. Následující výčet technologií není úplný, řada technologií byla záměrně vynechána, z důvodu nevhodnosti pro potřeby této práce. Například *Near-Field Communication* (NFC) a RFID z důvodu krátkého dosahu. Nebo 5G či Li-Fi, které v současné době nejsou rozšířené.

Technologie byly rozděleny do dvou celků podle jejich dosahu. Krátkou vzdáleností je v této práci myšlena vzdálenost mezi komunikujícími stranami menší než 1 km. Toto rozdělení je jenom orientační. Reálný dosah jednotlivých technologií se vlivem prostředí může i několikanásobně lišit od teoretických hodnot.

3.1 Bezdrátový přenos na krátkou vzdálenost

3.1.1 Bluetooth a BLE

Bluetooth je technologie určená k propojení několika zařízení v rámci *Personal Area Network* (PAN) sítě. Funguje v frekvenčním rozmezí 2,400 - 2,483 GHz. Hodí se pro zasílání menšího množství dat na krátkou vzdálenost. Dosah této technologie je typicky menší než 10 metrů, ale v závislosti na použité verzi a vysílači to může být až několik set metrů, v případě směrové antény i přes kilometr [32]. Používá se například k propojení příslušenství k telefonu nebo počítači. V průmyslu má využití pro různé detektory a sondy. V IoT pro chytré žárovky, termostaty, senzory atd. V standardu Bluetooth 4.0 byl představen *Bluetooth Low Energy* (BLE), zaměřující se na energetickou úsporu a malé množství přenášených dat na větší vzdálenost.

Bezpečnost

Během komunikace Bluetooth využívá pseudonáhodné přeskakování mezi kmitočty. Klíče jsou vygenerovány z *Personal Identification Number* (PIN) kódu, ten u omezených zařízení může být předkonfigurován. Autentizace se provádí pomocí 128 bitového klíče. Důvěrnost komunikace zajišťuje šifra SAFER v proudovém režimu s klíčem o maximální velikosti 128 bitů [33].

3.1.2 WiFi

Rodina standardů pro bezdrátovou komunikaci spadajících pod *Wireless local area network* (WLAN). Funguje na 2.4 GHz, ale různé verze využívají i jiná pásma jako 5 GHz (a), 6 GHz (ac), 60 GHz (ad) a 900 MHz (ah) [34]. Různé verze se vyznačují různými vlastnostmi, obecně platí, že čím vyšší frekvence, tím vyšší přenosová rychlost, ale zároveň menší dosah. Například 5 GHz má většinou vyšší rychlost a menší

dosah než 2.4 GHz. Pro IoT se velice hodí WiFi HaLow (ah), fungující na frekvencích okolo 900 MHz, vyznačující se dlouhým dosahem (1 km), nízkými energetickými nároky a schopností penetrovat silné zdi [35].

Bezpečnost

Bezpečnost je v sítích WiFi zajišťována různými opatřeními a jejich kombinacemi. K těmto například patří skrytí *Service Set IDentifier* (SSID), filtrování *media access control* (MAC) adres a zejména použití Wi-Fi security. To se skládá z protokolů WEP, WPA, WPA2 a WPA3 [36]. První zmíněný je z dnešního pohledu nebezpečný a neměl by se používat. WPA je náhrada za WEP, ale přejímá z něj některé nebezpečné vlastnosti, proto také nebývá příliš doporučován. V současnosti nejrozšířenějším je WPA2 [37], který se autentizuje k přístupovému bodu pomocí 4-way Handshake, s ním si také domluví šifrovací klíč odvozený z předsdíleného klíče. K šifrování samotné komunikace je poté možné použít AES nebo *Temporal Key Integrity Protocol* (TKIP). Nejnovější protokol je WPA3, který je považován za nejbezpečnější. V současné době však není tak rozšířený.

3.1.3 Zigbee

Bezdrátová technologie typu PAN, určena k přenosům malého množství dat. Cílí na automatizaci budov, průmyslovou automatizaci, dálkové ovládání elektrospotřebičů atp. Využívá pásmo 2.4 GHz. Má dosah až 75 metrů, využívá tzv. multiskokové ad-hoc směrování, které umožňuji tuto vzdálenost navýšit [38].

Bezpečnost

Bezpečnost je zajištěna pomocí black/white listů hardwarových adres, číslováním rámců a šifrováním komunikace. Šifrování může probíhat různými způsoby, nejběžnější je takový, že po žádosti zařízení o přístup do sítě brána zašle network key, který je zašifrován pomocí link key. Link key je vytvořen z tajemství, které znají obě strany. Následná komunikace se šifruje s AES128 za použití vypočteného network key [39].

3.1.4 Z-Wave

Z-wave je komunikační protokol vhodný k budování IoT sítí. Cílí na domácí automatizaci, budování chytrých domů a chytré spotřebiče (žaluzie, okna, dveře, termostaty atd.). Na rozdíl od předešlých nepoužívá přetížené pásmo 2,4 GHz, ale bezlicenční, méně vytížené 868,4 až 956 MHz. Typická vzdálenost mezi zařízeními je zhruba 30 metrů. Jednotlivá zařízení mezi sebou tvoří mesh síť, tím jsou schopny až několikanásobně prodloužit vzdálenost od centrálního bodu [40].

Bezpečnost

Do roku 2016 byla bezpečnost této technologie slabá. Od tohoto roku musí každé Z-wave zařízení podporovat Security 2 (S2) framework, který podstatně zabezpečení zlepšuje [41]. Šifrování se provádí na aplikační vrstvě. Každé zařízení má unikátní klíč, který se využívá k ustanovení sdíleného klíče v protokolu ECDH. Získaný klíč je následně použit k šifrování prostřednictvím AES128 [42].

3.2 Bezdrátový přenos na dlouhou vzdálenost

3.2.1 LoRa

Long-Range (LoRa) je technologie řadící se mezi *Low Power Wide Area Network* (LPWAN). V Evropě funguje v pásmu 868 - 870 MHz. Cílem této technologie je minimální spotřeba, malý datový tok a velký dosah. Zařízení využívající tuto technologii často vysílají pouze několikrát za den, a tím mohou mít životnost baterie i v řádu několika let. Reálný dosah je 10 km [43].

Bezpečnost

Každé LoRa zařízení má předinstalovaný jedinečný 64 b identifikátor a 128 b dlouhý klíč. K šifrování se používá AES. Integrita a autentičnost je zajištěna pomocí *Cipher-based Message Authentication Code* (CMAC), důvěrnost a ochrana proti replay útoku šifrováním v módu *Counter* (CTR) [44].

3.2.2 Sigfox

Sigfox je alternativou k technologii LoRa. Obě používají stejné pásmo, stejně jsou zaměřeny na IoT, nízkou spotřebu a velký dosah. Technologie se zejména liší tím, že Sigfox je operátor a pro použití jeho služeb je nutné, aby se v dané lokalitě vyskytoval. Dále Sigfox má obvykle menší energetickou spotřebu a menší přenosovou rychlost [43].

Bezpečnost

Šifrování probíhá podobně jako v sítích LoRa. Každé zařízení má z výroby unikátní symetrický klíč a identifikátor. Každá zpráva obsahuje kryptografický token, vypočítaný mimo jiné z klíče, *message authentication code* (MAC) a čísla paketu. Šifrování samotných dat se provádí klíčem odvozeným od device ID a Network Authentication Key (NAK). Je možné implementovat i vlastní řešení. Šifrování mezi základnovou stanicí a koncovým serverem je prostřednictvím VPN tenulů [45].

4 Příprava vývojového prostředí

Po zvážení všech možných postupů zprovoznění vývojového prostředí, byl zvolen postup využívající virtualizaci. Konkrétně manuální instalace všech potřebných programů do operačního systému Ubuntu, virtualizovaného programem VirtualBox z hostujícího počítače s Windows 10. Hlavní výhody této metody:

- Možnost vývoje z OS Windows.
- Jednoduché obnovení systému v případě jeho nefunkčnosti (pomocí snapshotů).
- Virtuální stroj je určen pouze k vývoji aplikací, vylučuje nechtěná interakce s jinými programy.
- Přenositelnost celého vývojového prostředí na jiný počítač, pomocí exportu VM.

Hlavní nevýhody:

- Nutnost přemosťování komunikačních rozhraní ze skutečného do virtuálního stroje.
- Pomalejší běh virtualizovaného počítače.

Vývojové prostředí

VirtualBox je open-source nástroj umožňující virtuální běh operačního systému a následné připojení *Universal Serial Bus* (USB) zařízení z hostitelského systému do virtuálního stroje [46]. Je možné jej zdarma stáhnout na stránkách výrobce [47]. Aby správně fungovalo přemosťování USB zařízení, je nutné ze stejné stránky stáhnout a nainstalovat Extension Pack. V této práci byla použita verze 6.1. Následně je možné stáhnout a nainstalovat Ubuntu [48]. Tato Linuxová distribuce byla zvolena z důvodu doporučení tvůrci RIOT. Použitá verze byla 20.04.

Na čisté Ubuntu je třeba doinstalovat všechny potřebné balíky pro vývoj (tzv. **dependencies** - nutné k správné funkčnosti RIOT). To se provede spuštěním následujícího příkazu v terminálu. Jedná se o různé nástroje nutné ke kompilaci, např. make, automake a také programovací jazyk python pro sériovou komunikaci atd.

```
sudo apt-get install build-essential git make automake
gcc curl unzip wget pkg-configautoconf libtool libusb-dev
libusb-1.0-0-dev libhidapi-dev libftdi-dev g++-multilib
gcc-multilib cppcheck doxygen graphviz pcregrep python
python3 python3-flake8 python3-serial
```

Nyní je možné stáhnout samotný **RIOT**, to se realizuje příkazem níže.

```
git clone git://github.com/RIOT-OS/RIOT.git
```

Dalším krokem je nainstalovat **textový editor** pro pohodlnou editaci kódu. Ten může být libovolný. Následující příkaz nainstaluje Sublime Text.

```
sudo snap install sublime-text --classic
```

V této fázi je již možné vytvořit první aplikaci a otestovat jí na RIOT native platformě. RIOT má v sobě připravené různé aplikace, které je možné použít na otestování funkčnosti systému. Nachází se ve složkách examples a tests. Do jedné z nich je možné se přesunout pomocí následujícího příkazu.

```
cd RIOT/examples/hello-world/
```

Zkrácený obsah souborů nacházejících se v této složce je možno vidět na výpisech 4.1 a 4.2. Zavoláním následujícího příkazu dojde ke kompilaci a nahrání kódu do virtuální desky native. Tím je možné zkontrolovat, že vše funguje správně.

```
make flash term
```

Výsledek tohoto kroku lze vidět na obrázku 4.1, zobrazující výpis z terminálu, potvrzující funkčnost RIOT.

```
main(): This is RIOT! (Version: 2020.07)
Hello World!
You are running RIOT on a(n) native board.
This board features a(n) native MCU.
```

Obr. 4.1: Výpis z terminálu pro desku native

Výpis 4.1: Soubor main.c

```
#include <stdio.h>                                                    1
int main(void){                                                       2
 puts("Hello␣World!");                                                3
 printf("You␣are␣running␣RIOT␣on␣a(n)␣%s␣board.\n"                    4
    , RIOT_BOARD);                                                    5
 printf("This␣board␣features␣a(n)␣%s␣MCU.\n", RIOT_MCU);              6
 return 0;}                                                           7
```

Výpis 4.2: Soubor Makefile

```
APPLICATION = hello-world                                             1
BOARD ?= native                                                       2
RIOTBASE ?= $(CURDIR)/../..                                           3
include $(RIOTBASE)/Makefile.include                                  4
```

Toolchain

Pokud by se v této fázi vybrala cílová deska jiná než native, došlo by k chybě. Je totiž nutné doinstalovat toolchain pro danou desku. Každý toolchain obsahuje jiné programy, instalace jenom některých z nich může způsobit nepředvídatelné chování. Bohužel dokumentace je pro některé desky velmi stručná, a tak je složité dohledat přesně jaký software se má doinstalovat. Zprovoznění některých desek je otázkou jednoho příkazu a u některých mnohem více.

Pro přidání podpory desky **Arduino Nano** stačí do terminálu napsat následující příkaz.

```
sudo apt-get install gcc-avr avr-libc avrdude
```

Tím se nainstaluje kompilátor jazyka C pro architekturu AVR, nainstalují se potřebné knihovny a nainstaluje se nástroj avrdude, sloužící k nahrávání kódu do zařízení.

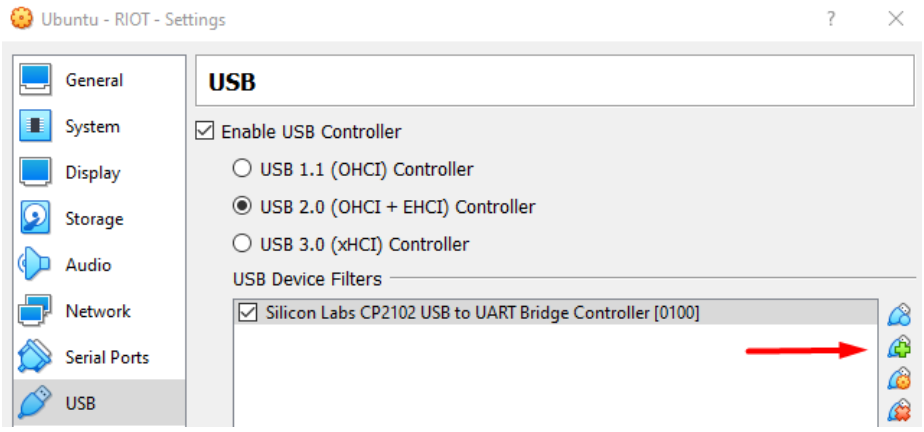
Toolchain pro desky založené na **ESP8266** se nainstaluje níže uvedenými příkazy. V tomto případě byla dokumentace kvalitní [49]. Exportované cesty jsou nastaveny pro uživatele ubuntu, v případě jiného uživatele je nutné je přizpůsobit.

```
mkdir -p $HOME/esp
cd $HOME/esp
git clone https://github.com/gschorcht/xtensa-esp8266-elf
git clone https://github.com/gschorcht/RIOT-Xtensa-ESP826
6-RTOS-SDK.git ESP8266_RTOS_SDK
cd ESP8266_RTOS_SDK/
git checkout release/v3.1-for-riot-os-v2
export PATH=/home/ubuntu/esp/xtensa-esp8266-elf/bin:$PATH
export ESP8266_RTOS_SDK_DIR=/home/ubuntu/esp/ESP8266_RTOS
_SDK
```

Flash a sériová komunikace

Pro nahrání kódu na zařízení je nutné přemostit rozhraní, ke kterému je zařízení připojeno. To se provede ve VirtualBoxu v sekci `nastavení` virtuálního stroje, záložka `USB`. Zde kliknutím na tlačítko s ikonou plus je možné vybrat zařízení k přemostění. Výsledek tohoto kroku je vidět na obrázku 4.2. Po zapnutí virtuálního stroje by ve složce `/dev/` mělo přibýt přemostěné zařízení. Pokud se tak nestalo, je nutné zařízení odpojit a zase připojit. Jestliže se ani poté neobjevuje, je třeba Ubuntu vypnout a ve stejné záložce vybrat přepínač `USB 3.0 (xHCI) Controller`.

Nyní nic nebrání nahrání kódu do připojeného zařízení. Provede se to vysláním následujícího příkazu ze složky projektu.



Obr. 4.2: Přemostění USB rozhraní

```
make flash term BOARD=esp8266-esp-12x
```

Pro nahrání kódu do zařízení Arduino Nano je nutné změnit proměnnou BOARD následovně: `BOARD=arduino-nano`. Oba příklady předpokládají, že zařízení je připojené jako `ttyUSB0`. Pokud je deska připojena k jinému rozhraní, tak je to nutné do příkazu přidat (např. `PORT=/dev/ttyUSB1`).

Slovo term v příkazu říká, že po nahrání kódu má začít sériová komunikace s vývojovou deskou. V obou případech se automaticky zavolá nástroj `pyterm` a předají se mu parametry, se kterými se má komunikace zahájit (port, boundrate, atd.). Výsledek nahrání kódu a zahájení sériové komunikace je vidět na obrázku 4.3.

Při nahrávání kódu se může objevit chyba: `Permission denied`, ta je způsobena nedostatečným oprávněním pro přístup k USB zařízení. Chybu lze opravit následujícím příkazem.

```
sudo chmod a+rw /dev/ttyUSB0
```

```
2020-12-09 10:40:56,224 # main(): This is RIOT! (Version: 2020.07)
2020-12-09 10:40:56,225 # Hello World!
2020-12-09 10:40:56,230 # You are running RIOT on a(n) esp8266-esp-12x board.
2020-12-09 10:40:56,234 # This board features a(n) esp8266 MCU.
```

Obr. 4.3: Výpis z terminálu pro desku ESP8266

Návod v této kapitole popisuje zprovoznění vývojového prostředí pro zařízení Arduino Nano a ESP8266. Detailnější popis celého procesu je možné zhlédnout na vytvořeném videonávodu [50][51]. V příloze je také detailní psaný návod, kde je celý postup popsán i pro ostatní použité desky (ESP32, Arduino UNO).

5 Výkonnostní testy na různých platformách

Výkonnostní testy byly prováděny na následujících zařízeních: Arduino Nano, Arduino UNO, ESP8266, ESP32 a STM32F103C8T6. Tato zařízení byla vybrána pro jejich podporu ze strany RIOT a relativní dostupnosti. Jedná se o desky různých výrobců a různých architektur. Zásadně se také liší paměťovými a výkonnostními možnostmi. To umožňuje otestovat chování RIOT OS v různých podmínkách. Kód použitý k testování výkonu se ve většině případů mezi deskami nelišil. Bohužel někdy jej bylo nutné modifikovat pro dané zařízení (například z důvodu nedostatečné paměti desky). Během testování bylo zjištěno, že výsledky desek Arduino NANO a UNO jsou zaměnitelné. Proto jsou ve výsledcích prezentovány pouze časy modelu NANO. Testovaná zařízení lze vidět na obrázku 5.1 a jejich porovnání v tabulce 5.1.

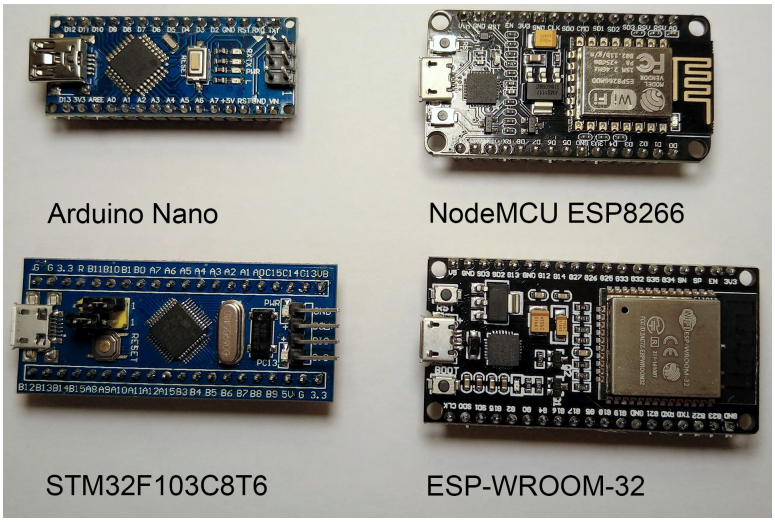
Měření bylo prováděno metodou odečtení počátečního času (začátek operace) a koncového času potřebného k výpočtu. Časy byly zaznamenávány v milisekundách, to je maximální přesnost, při které je schopen nástroj pyterm pracovat. Do výsledku je započítán i čas potřebný k odeslání a zobrazení zprávy o začátku a konci testu. To mohlo mít vliv na výsledný čas. Proto výsledky v jednotkách milisekund mohou být nepřesné. U většiny testů byly naměřené časy v desítkách či stovkách milisekund. U nich je tato nepřesnost zanedbatelná.

Většina testů proběhla v RIOT verzi 2020.10. Během testování se zjistilo, že některé funkce nefungují správně [52], proto některé testy proběhly ve starší verzi 2020.07. Kontrolním měřením stejného kódu na stejném zařízením s různými verzemi RIOT bylo zjištěno, že se časy nijak neliší a je tedy možné výsledky i tak porovnat.

Výsledné hodnoty v tabulkách jsou vypočítány z průměru deseti naměřených časů. Pokud není uvedeno jinak, tak uvedená hodnota v tabulce/grafu je v milisekundách. Použité kódy jednotlivých testů se nacházejí v příloze.

Arduino Nano

Zástupce platformy ATmega architektury AVR. Jedná se o miniaturní vývojovou desku s 8 bitovým mikrokontrolérem Atmega328p. Obsahuje 14 digitálních I/O pinů, 6 z nich podporuje *Pulse-width modulation* (PWM). Dále 6 analogových input pinů vhodných třeba ke sběru dat z čidel. Kompletní specifikaci si je možné přečíst v přiloženém zdroji [53]. Tato deska je velice populární a snadno k dostání. Její licencování umožňuje vznik klonů, které se vyznačují velmi nízkou cenou a často nekvalitním provedením. Deska má zabudovaný USB to *Transistor-Transistor Logic* (TTL) serial chip, takže k jejímu programování a následnou komunikaci není potřeba žádný další hardware.



Obr. 5.1: Použité vývojové desky

Tab. 5.1: Přehled vývojových desek

	Arduino Nano	**ESP8266**
mikrokontrolér	atmega328p 8bit	Tensilica Xtensa LX106 RISC 32 bit
architektura	AVR	ESP
rychlost procesoru	16 MHz	80 MHz
flash paměť	32 KB	4 MB
RAM paměť	2 KB	64 KB
orientační cena	100 - 400 Kč	100 - 200 Kč
	ESP32	**STM32F103C8T6**
mikrokontrolér	Tensilica Xtensa LX6 32 bit	ARM Cortex-M3
architektura	ESP	ARM
rychlost procesoru	160 - 240 MHz	72 MHz
flash paměť	4 MB	64/128 KB
RAM paměť	520 KB	20 KB
orientační cena	200 - 500 Kč	100 - 200 Kč

ESP8266

MCU firmy Espressif Systems se prodává v několika variantách lišících se zejména počtem *General-purpose input/output* (GPIO) pinů a přítomností *Analog-to-digital converter* (ADC). Samotný čip je umístěný na vlastní desce. Tento modul (čip na *Printed circuit board* (PCB)) poté další výrobci integrují do větších celků přidávajících další funkcionality [54]. Těmi zejména jsou regulátor napětí a micro-USB

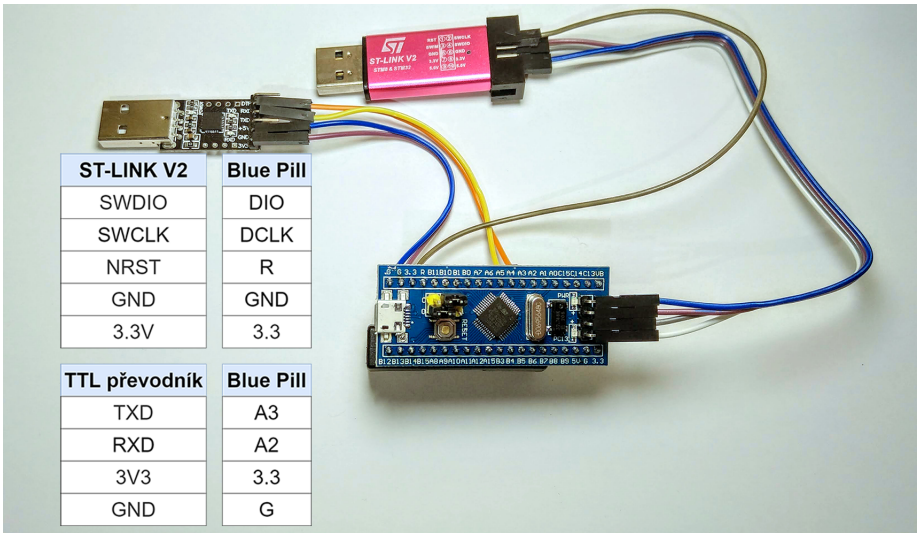
konektor. Dále například USB-TTL převodník pro snadné programování. Velká výhoda tohoto modulu je podpora WiFi, a tak možnost bezdrátového připojení bez dalšího potřebného hardwaru. Pro výkonnostní testy byl použit NodeMCU ESP8266 integrující modul ESP-12E, mající všechny zmíněné výhody. K programování tedy nebylo třeba žádné další zařízení.

ESP32

Tato deska má všechny funkcionality jako ESP8266 a přidává k nim řadu dalších. Například silnější dvoujádrový procesor, větší počet GPIO pinů, více RAM, bluetooth, Hallův senzor, 12 bitový ADC a další [55][56]. Taktéž jako předchozí se vyskytuje ve formě modulů, které výrobci integrují do svých desek. K testům byla zvolena verze ESP-WROOM-32, přidávající mimo jiné převodník pro sériovou komunikaci, a tím usnadňuje programování. RIOT v současné době podporuje pouze jedno jádro tohoto procesoru.

STM32F103C8T6 (Blue Pill)

Výkonná vývojová deska firmy STMicroelectronics využívá jádro ARM Cortex-M3. Z důvodu jejího lidsky nepřívětivého názvu se pro ní ustálilo také označení Blue Pill. Na desce se nachází micro-USB konektor a regulátor napětí pro snadné napájení [57]. Na rozdíl od desek ESP nepodporuje WiFi a v zařízení se také nenachází USB-TTL převodník. K programování a následné komunikaci je tedy nutné další zařízení. Programování zajišťovalo zařízení ST-Link V2 a následnou komunikaci TTL převodník s CP2102. Dalo by se to provést i jiným způsobem, avšak tento postup je vývojáři RIOT doporučován [58]. Zapojení je možno vidět na obrázku 5.2.



Obr. 5.2: Blue Pill programování

5.1 Modulární aritmetika

Výkonnostní test modulární aritmetiky proběhl pomocí knihovny micro-ecc. Knihovnu je nutné zkompilovat s proměnnou `uECC_ENABLE_VLI_API`, jinak operace nejsou dostupné. Knihovna umožňuje modulární operace pouze pro čísla o stejné velikosti jako podporované křivky. Jedná se tedy o čísla délky 160, 192, 224 a 256 bitů. Byly otestovány všechny podporované délky. Na desce Arduino Nano kód nebylo možné spustit, proto se ve výsledcích nevyskytuje. Byly testovány celkem tři operace: modulární redukce, modulární sčítání a násobení. Výsledky měření lze vidět v tabulce 5.2. Některé výpočty byly tak rychlé, že je nebylo možné použitou metodou změřit. Ve výsledcích jsou označeny jako <1.

Všechny naměřené časy byly velice rychlé, nejpomalejším zařízením v testu se ukázalo STM32F103C8T6. Desky ESP32 a ESP8266 dosahovaly podobných výsledků. Z testů vyplývá, že nejnáročnější testovanou operací bylo modulární násobení.

Tab. 5.2: Výkonnostní testy - modulární aritmetika

	Modulární redukce [ms]			
	160b	192b	224b	256b
ESP8266	1	<1	1	1
ESP32	1	<1	1	1
STM32F103C8T6	2	2	3	3
	Modulární sčítání [ms]			
	160b	192b	224b	256b
ESP8266	1	<1	<1	1
ESP32	1	1	<1	<1
STM32F103C8T6	1	1	<1	<1
	Modulární násobení [ms]			
	160b	192b	224b	256b
ESP8266	1	1	1	2
ESP32	2	<1	1	1
STM32F103C8T6	2	2	2	3

5.2 Symetrická kryptografie

Testy symetrické kryptografie byly provedeny knihovnou tinycrypt a RIOT modulem Crypto. Byla vybrána šifra AES128, protože ji implementují obě knihovny. Je tedy možné porovnat výkon nejen mezi zařízeními, ale také celými knihovnami. Byly otestovány dva módy šifry, *Electronic Code Book* (ECB) a *Counter with cipher block chaining message authentication code* (CCM). Testování probíhalo následovně. Byl změřen čas potřebný k zašifrování 1024 a 2048 AES bloků, tedy 131072 b, resp. 262144 b dat. Z tohoto času byla vypočtena bitová rychlost, tj. kolik bloků (přepočteno na bity) by dané zařízení zašifrovalo za jednu sekundu. Výsledná rychlost je průměrem obou vypočtených hodnot. Proces se opakoval pro dešifrování. Podobně probíhalo i testování módu CCM, kdy se šifře předkládaly různě dlouhé vstupy. Test nebyl proveden na zařízení Arduino Nano, protože se kód ani po zkrácení na minimum do paměti této desky nevejde a knihovnu tinycrypt není možné zkompilovat pro 8 bitovou architekturu.

Všechny naměřené, respektive vypočtené, hodnoty lze vidět v tabulce 5.3. Z výsledků vyplývá, že modul crypto je podstatně rychlejší při šifrování i dešifrování než knihovna tinycrypt. Například zařízení ESP32 šifrovalo rychlostí 5,09 Mbps pomocí modulu crypto oproti 0,8 Mbps s knihovnou tinycrypt. To je více než šestinásobný rozdíl. Nejrychlejším zařízením v testu bylo ESP32, které svými výsledky převyšovalo ostatní desky. CCM mód je podstatně výpočetně náročnější, proto se očekávalo, že jeho výsledky budou minimálně dvakrát pomalejší (při šifrování i dešifrování se šifra aplikuje dvakrát a navíc využívá řetězení bloků). To se také potvrdilo, většina výsledků byla několikanásobně horší. Zajímavé zjištění je, že šifrování je bez použití módů podstatně rychlejší než dešifrování a naopak v módu CCM jsou tyto operace stejně náročné.

Tab. 5.3: Výkonnostní testy - AES

AES128	**crypto [Mbps]**		**tinycrypt [Mbps]**	
	šifrování	dešifrování	šifrování	dešifrování
ESP8266	1,44	0,42	0,43	0,17
ESP32	5,09	2,66	0,8	0,18
STM32F103C8T6	3,07	1,48	0,68	0,16
AES-CCM-128	**crypto [Mbps]**		**tinycrypt [Mbps]**	
	šifrování	dešifrování	šifrování	dešifrování
ESP8266	0,43	0,43	0,07	0,07
ESP32	1,43	1,41	0,16	0,16
STM32F103C8T6	0,83	0,84	0,13	0,13

5.3 Hašovací funkce

Testování probíhalo pomocí knihovny tinycrypt a RIOT modulu hashes. Byla vybrána funkce SHA256. Měření probíhalo podobně jako u blokové šifry AES128. Byl sledován čas potřebný k výpočtu 1024 a 2048 haší. Vstupní řetězec byl fixní délky 256 bitů. Následně byla vypočtena průměrná rychlost obou pokusů.

Výkonnostní test nebylo možné provést na desce Arduino Nano pro knihovnu tinycrypt. Ze stejného důvodu jako při testu AES128, tedy nekompatibilní architektura.

Naměřené hodnoty si lze prohlédnout v tabulce 5.4. V tomto testu se naopak ukázala výkonnější knihovna tinycrypt, která dosahovala lepších časů než RIOT modul hashes, tentokrát však rozdíl nebyl tak výrazný. Výsledky ukázaly velký rozdíl výkonu mezi deskou Arduino Nano a ostatními. Konkrétně deska Arduino dosáhla více než stokrát horšího výsledku než druhá nejpomalejší. Nejrychlejší zařízení bylo opět ESP32, avšak v tomto testu byl náskok menší.

Tab. 5.4: Výkonnostní testy - SHA256

SHA256	**crypto [Mbps]**	**tinycrypt [Mbps]**
Arduino Nano	0,02	nelze
ESP8266	2,46	2,75
ESP32	2,88	3,26
STM32F103C8T6	2,21	2,53

5.4 Eliptické křivky

Měření operací s body nad eliptickými křivkami bylo prováděno pomocí knihovny micro-ecc. Byly měřeny dvě základní operace. Násobení bodu na eliptické křivce skalárem a sečtení dvou bodů na eliptické křivce. Operace násobení je implementovaná ve funkci `uECC_point_mult()`. Tato funkce a řada dalších není uživateli hned dostupná. Je třeba v souboru `Makefile` definovat hodnotu `uECC_ENABLE_VLI_API`, tím se tyto funkce zpřístupní. Měření operace sčítání je složitější, protože knihovna přímo tuto funkci nenabízí, přestože tuto operaci provádí například v algoritmu ECDSA. Sčítání bylo tedy zpřístupněno modifikací knihovny, konkrétně přidáním funkce `uECC_point_add()` [61]. Výsledné časy testů sčítání bodů a násobení bodu skalárem jsou zobrazeny v tabulce 5.5. Tyto hodnoty byly dále vykresleny do grafů. Na obrázku 5.3 je vidět graf násobení bodu skalárem (na ose y jsou uvedeny sekundy) a na obrázku 5.4 je graf vykreslující výsledky testu sčítání bodů.

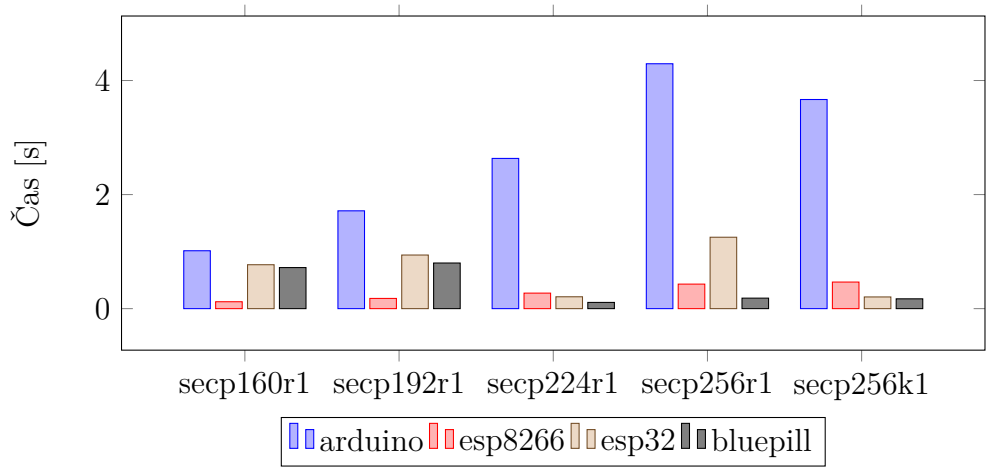
Zařízení Arduino Nano nemá dostatečnou paměť na běh celých testů. Je nutné zakomentovat některou ze křivek (netestovat všechny křivky najednou). Poté kompilace proběhne v pořádku.

Z naměřených dat jasně vychází, že operace násobení je podstatně výkonově náročnější než operace sčítání. Deska Arduino Nano se podle očekávání umístila na posledním místě s časy zhruba desetkrát pomalejšími než ostatní. Čas potřebný k vynásobení bodu skalárem byl u této desky okolo čtyř sekund, což je pro praktickou implementaci této operace nepřijatelné. Operace sčítání trvala u většiny desek okolo tří milisekund. Také deska Arduino s výsledky maximálně 41 milisekund je pro tuto operaci použitelná. V tomto testu byla nejrychlejší deska STM32F103C8T6, která i se svým pomalejším procesorem porazila výkonnější desku ESP32.

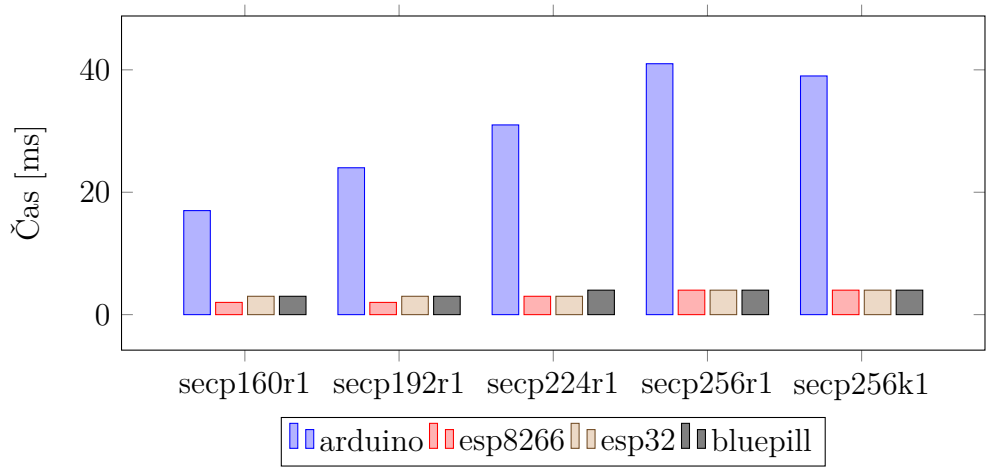
Tab. 5.5: Výkonnostní testy - násobení a sčítání na eliptické křivce

	Násobení bodu skalárem [ms]				
	secp160r1	secp192r1	secp224r1	secp256r1	secp256k1
Arduino Nano	1015	1715	2634	4294	3667
ESP8266	121	179	272	430	466
ESP32	77	94	208	252	205
STM32F103C8T6	72	80	109	184	172
	Sčítání bodů [ms]				
	secp160r1	secp192r1	secp224r1	secp256r1	secp256k1
Arduino Nano	17	24	31	41	39
ESP8266	2	2	3	4	4
ESP32	3	3	3	4	4
STM32F103C8T6	3	3	4	4	4

Obr. 5.3: Graf výkonnostního testu násobení bodu skalárem



Obr. 5.4: Graf výkonnostního testu sčítání bodů



5.5 ECDH a ECDSA

Měření těchto algoritmů probíhalo opět pomocí knihovny micro-ecc. Použitý kód byl `test_ecdh.c` a `test_ecdsa.c` dostupný přímo v knihovně. Pro potřeby výkonnostních testů byl mírně modifikován. U algoritmu ECDH se měřil čas potřebný k vytvoření dvou páru klíčů a výpočtu dvou (totožných) sdílených klíčů. Jednalo se tedy o ustanovení klíčů mezi dvěma entitami v rámci jednoho zařízení. Kontrola správnosti výpočtu do času není zahrnuta, protože byla prováděna mimo sledovaný úsek. Výsledné časy jsou v tabulce 5.6. Do výsledného času algoritmu ECDSA je započítáno vytvoření dvojice klíčů, samotný podpis a jeho verifikace. Naměřené hodnoty jsou taktéž v tabulce 5.6. Graf výsledků algoritmu ECDH je na obrázku 5.5. Graf algoritmu ECDSA je na obrázku 5.6.

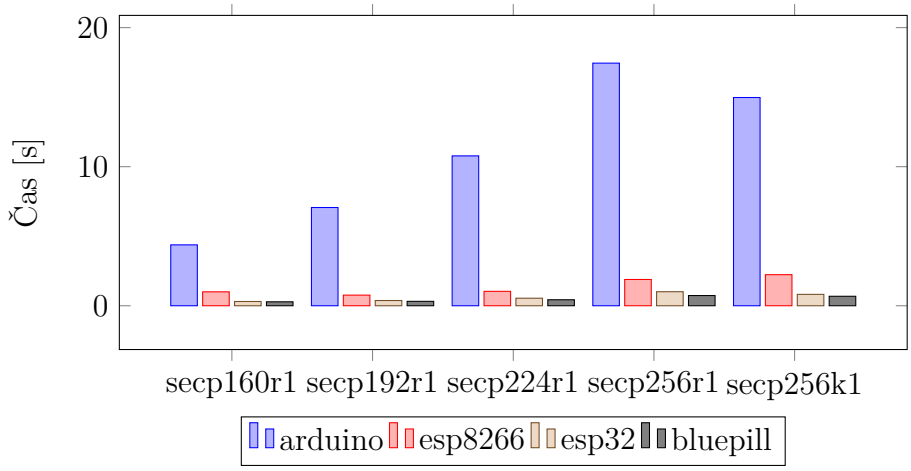
Při testování zařízení Arduino Nano docházelo k nestandardnímu chování. I přes úspěšnou kompilaci a nahrání kódu docházelo k pádům systému, které znemožňovaly měření. Tyto problémy se nepodařilo vyřešit, proto se přistoupilo řešení, že časy pro desku Arduino nano v této sekci nebyly naměřeny pomocí RIOT OS. Místo toho byl kód nahrán na zařízení pomocí Arduino IDE. Jednalo se o totožný kód s drobnými úpravami. Úpravy spočívaly zejména ve změně způsobu výpisů do výstupu a v přidání sekce setup, povinné pro úspěšnou kompilaci z Arduino IDE. Výsledky tedy nezohledňují možný dopad režie RIOT OS na výkon. Je však nepravděpodobné, že by se režie jednoho systému lišila od druhého takovým způsobem, že by se to významně projevilo v testech, proto se i přes to deska Arduino Nano do porovnání zařadila.

Výsledky testů ECDH a ECDSA ukazují, že Arduino Nano není vhodné k implementaci těchto protokolů. Taktéž zařízení ESP8266 v testu ECDH dosahovalo poměrně vysokých hodnot, které by byly v praktické implementaci problematické a pro křivku secp256r1 nepřijatelné. Nejrychlejším zařízením bylo opět STM32F103C8T6, které znovu překonalo hardwarově vybavenější ESP32.

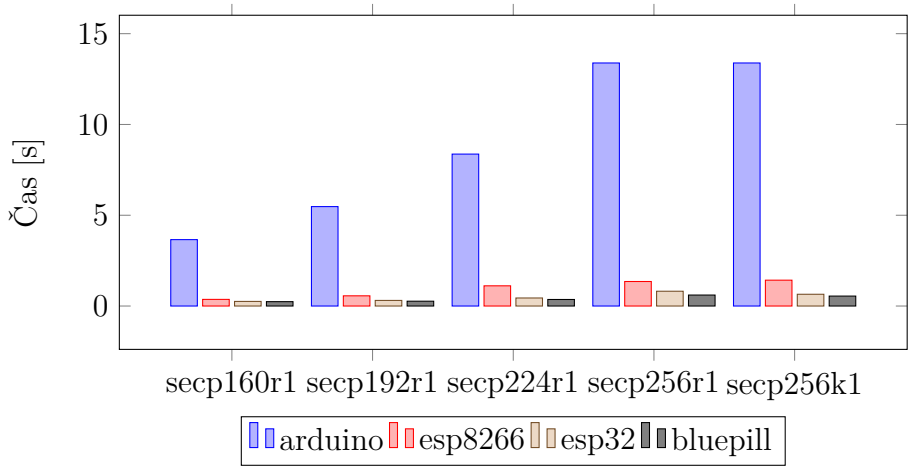
Tab. 5.6: Výkonnostní testy - ECDH a ECDSA

	ECDH [ms]				
	secp160r1	secp192r1	secp224r1	secp256r1	secp256k1
Arduino Nano	4379	7065	10778	17778	14975
ESP8266	1001	764	1038	11889	2234
ESP32	306	376	542	1009	819
STM32F103C8T6	284	317	431	734	682
	ECDSA [ms]				
	secp160r1	secp192r1	secp224r1	secp256r1	secp256k1
Arduino Nano	3657	5476	8368	13386	11266
ESP8266	366	562	1113	1351	1424
ESP32	254	309	443	815	648
STM32F103C8T6	241	266	361	605	548

Obr. 5.5: Graf výkonnostního testu ECDH



Obr. 5.6: Graf výkonnostního testu ECDSA

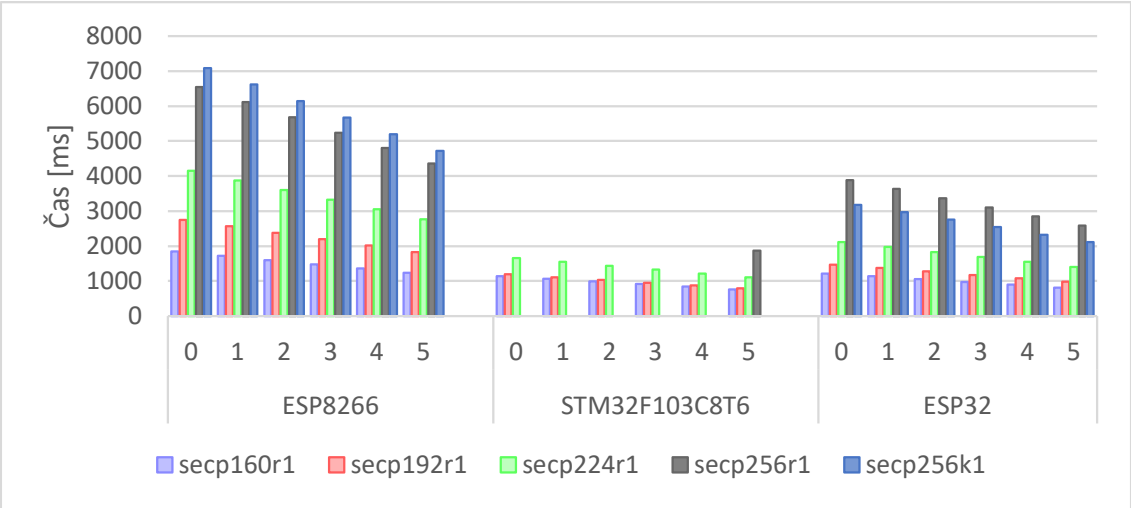


5.6 KVAC a RKVAC

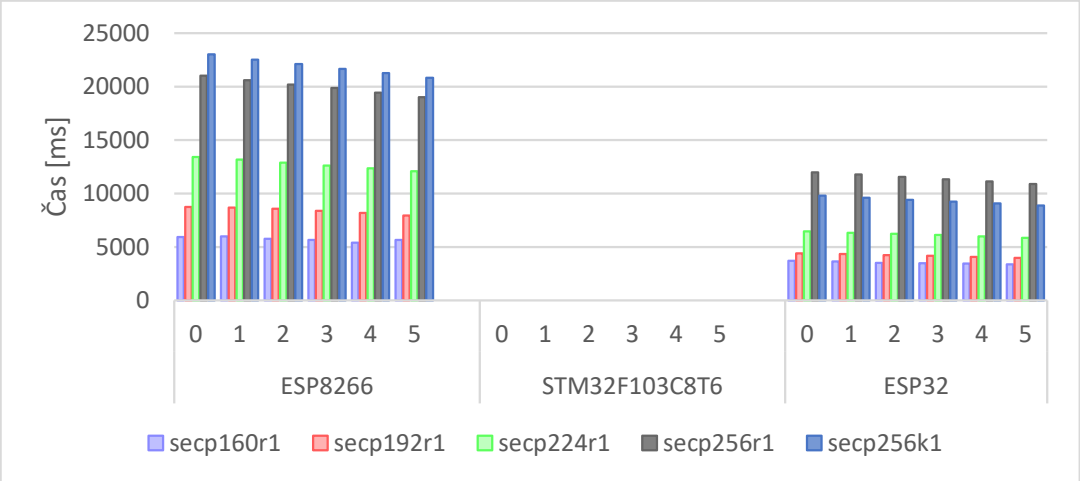
Poslední benchmarkový test se věnoval schématům *Keyed-Verification Anonymous Credentials* (KVAC) [59] a *Revocable KVAC* (RKVAC) [60]. KVAC schéma umožňuje anonymní autentizaci uživatele na základě atributů, kterými mohou být pohlaví, věk, jméno atd. Uživateli je umožněn přístup, pokud dokáže, že disponuje konkrétními atributy. RKVAC přidává revokační autoritu, která je schopna vyřadit uživatele ze systému [61]. Otestování výkonu těchto schémat bylo provedeno pomocí aplikací pana Cvrčka [61]. Z důvodu časové náročnosti tohoto testu jsou výsledné hodnoty průměrem pouze dvou naměřených hodnot. Zařízení Arduino Nano bylo z testu vynecháno, protože kód (pod RIOT OS) nelze na této desce spustit. Níže byly vybrány pouze dva grafy zastupující každé ze schémat. V příloze práce jsou další tabulky a grafy i pro jiné počty vydaných atributů. Na zařízení STM32F103C8T6 nebylo možné změřit schéma RKVAC.

Naměřené hodnoty schématu KVAC lze vidět na obrázku 5.7. Výsledné časy schématu RKVAC jsou zobrazeny na obrázku 5.8. V grafech je na ose x počet odhalených

atributů a na ose y je čas v milisekundách. Každá skupina sloupců představuje jedno zařízení v tomto pořadí: ESP8266, STM32F103C8T6 a ESP32. Naměřené hodnoty ukázaly, že zařízení ESP8266 nemá dostatečný výkon pro implementaci těchto schémat. V testech byly této desce naměřeny časy v řádech několika sekund, a to je pro použití nepřijatelné. Také deska ESP32 při použití bezpečné křivky secp256r1 a pěti vydaných atributů dosahovala ve schématu KVAC nepřijatelné časy v průměru přes tři sekundy. Schéma RKVAC bylo příliš náročné pro všechny měřené desky. Zařízení STM32F103C8T6 nebylo možné plně otestovat, ale i z neúplných výsledků vychází, že v testu bylo nejrychlejší. Na všech deskách byl průběh testování lineární.



Obr. 5.7: Graf výkonnostního testu KVAC - 5 vydaných atributů



Obr. 5.8: Graf výkonnostního testu RKVAC - 5 vydaných atributů

5.7 Zhodnocení výsledků

Z výsledků měření vyplývá, že deska **Arduino Nano** se z důvodu nízkého výkonu, malé paměti a 8 bitové architektuře, příliš nehodí k náročným kryptografickým operacím. Nejenom, že výsledné časy/rychlosti byly násobně pomalejší než ostatní desky, ale také samotné zprovoznění jednotlivých testů bylo často na této desce problematické či nemožné.

Deska **ESP8266** v testech vykazovala relativně dobré výsledky, zejména v hašování se téměř vyrovnala i výkonnějším deskám, ale v testu ECDH, za použití bezpečného 256 bitové klíče, byly výsledné časy pro praktickou implementaci nepřijatelné. Práce s touto deskou byla velice pohodlná, během testování se nevyskytl žádný problém.

Vývojová deska s **ESP32** byla papírově nejvýkonnějším zařízením v testu, avšak její výsledky tomu neodpovídaly. Jediným testem, kde tato deska dosahovala výraznějšího náskoku byl AES. Nutno podotknou, že testy nevyužívaly dvoujádrový procesor přítomný na této desce, protože RIOT v současnosti podporuje pouze jedno jádro. I za použití pouze jednoho jádra byl očekáván zhruba dvojnásobný nárůst výkonu oproti desce ESP8266, což se nestalo. Zprovoznění a programování této desky bylo stejně jako v případě předchozí bezproblémové.

Poslední testovanou deskou byla **STM32F103C8T6**, též bluepill. I přes pomalejší procesor se tato deska ukázala jako nejvýkonnější pro práci s knihovnou micro-ecc. Za svoji nízkou pořizovací cenu (z Číny okolo 40Kč) nabízí velmi vysoký poměr cena/výkon. Nevýhodou je relativně složité programování a sériová komunikace, a to z důvodu absence převodníku. Zprovoznění této desky je velice jednoduché, stačí pouze doinstalovat vhodný kompilátor. Programování není moc pohodlné, je nutné nejdříve přemostit boot pin, připojit programátor a následně ve správný okamžik stisknout tlačítko reset. Pro spuštění kódu je nutné vrátit jumper do původní pozice a zmáčknout znovu reset. Nedodržení tohoto postupu vede k chybě a je nutné celý proces opakovat. Během práce se občas vyskytl problém, a to že RIOT spadl, respektive vypsal hlášku kernel panic a restartoval se. Kromě tohoto problému bylo fungování víceméně bezproblémové.

6 Návrh a implementace systému

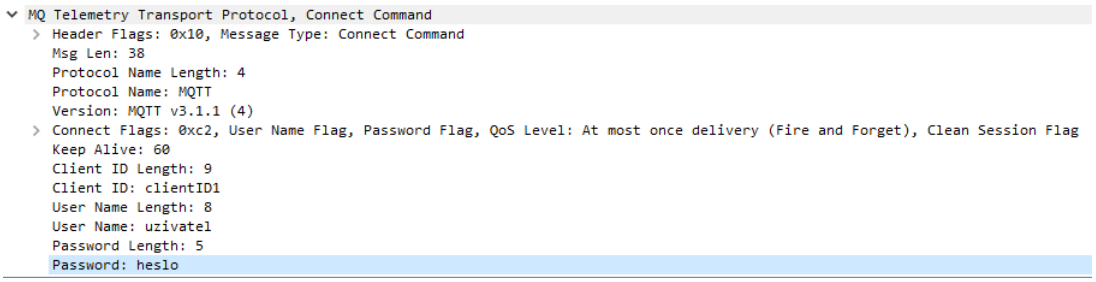
Následující kapitola popisuje architekturu a implementaci navrženého systému. Tento systém byl publikován v článku na konferenci student EEICT 2021 [63]. K pochopení fungování systému je nejprve nutné vysvětlit jednotlivé technologie, ze kterých se systém skládá. Bezdrátový přenos dat zajišťuje technologie WiFi, která byla popsána v kapitole 3, proto zde již nebude rozebírána. Výměnu zpráv mezi jednotlivými zařízeními zajišťuje komunikační protokol *Message Queuing Telemetry Transport* (MQTT). Na klientech běží operační systém RIOT, který byl popsán v kapitole 2. Zdrojový kód celého systému je v příloze práce.

MQTT

MQTT je jednoduchý, nenáročný a robustní komunikační protokol, určený pro IoT zařízení [62]. Vyznačuje se minimální režií, schopností fungovat na velmi omezených zařízeních a v nespolehlivých sítích s nízkou šířkou pásma a vysokou latencí. Je vhodný k přenosu krátkých zpráv. Zpravidla funguje nad transportním protokolem *Transmission Control Protocol* (TCP), ale existují i verze nad *User Datagram Protocol* (UDP) a Bluetooth. Řadí se k architekturám klient-server. Veškerá komunikace probíhá mezi klientem a serverem. Klienti mezi sebou komunikovat nemohou. MQTT server se nazývá `broker` Jeho hlavní úloha je směrování zpráv. Klienti mohou být dvojího typu. Klient typu `publisher` odesílá data brokeru a klient typu `subscriber` data od brokeru přijímá. MQTT zprávy se zasílají do tzv. témat (ang. topic). Jakmile publisher zašle zprávu do tématu, broker zprávu přepošle všem klientům (typu subscriber), kteří dané téma odebírají. Jeden klient může odebírat několik témat a zároveň i odesílat data do různých témat. Klient tedy může být současně publisher i subscriber. Témata jsou tvořena řetězci znaků. Pro lepší přehlednost se kombinují do stromové struktury. Např. `budova_A/patro_2/mistnost_5/klient_1`. Jednotlivá témata se oddělují lomítky. V takovém případě platí, že téma `klient_1` je podtématem tématu `mistnost_5`, atd. Každý klient má unikátní identifikátor (`Client ID`), který se používá k přihlášení k brokeru i ke směrování zpráv.

Protokol ve výchozím nastavení nezajišťuje žádné zabezpečení. Po připojení k brokeru může každý klient číst a zapisovat do jakéhokoliv tématu, a to je z hlediska bezpečnosti nepřijatelné. Proto byla do protokolu přidána autentizace pomocí jména a hesla. Před tím, než klient může posílat jakékoliv zprávy, musí se nejdříve přihlásit k brokeru. Žádost o přihlášení zajišťuje zpráva `CONNECT`. Tato zpráva, zachycená programem Wireshark, je vidět na obrázku 6.1. Kromě povinného pole Client ID, jsou zde i nepovinná pole User Name a Password, které mohou být použity k autentizaci. Broker může například vynutit, že povolí přihlášení pouze klientům s určitým jménem a heslem nebo umožní přihlášení všem, ale omezí čtení/zápis do určitých

témat na základě Client ID nebo jména a hesla. Tento mechanismus má bohužel podstatnou vadu. Všechny hodnoty (včetně hesla) se posílají v otevřené podobě a je tak možné zprávu zachytit a hodnoty jednoduše přečíst viz obrázek 6.1. Z tohoto důvodu je vždy nutné MQTT autentizaci kombinovat s dalšími metodami zabezpečení (šifrování komunikace) nebo zvolit metodu jinou.



Obr. 6.1: MQTT zpráva CONNECT zachycená programem Wireshark

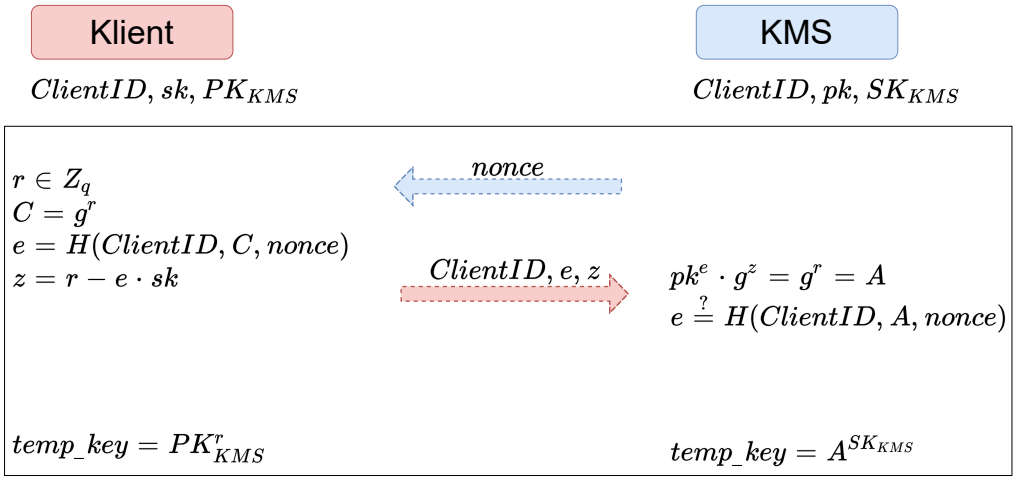
Existují různé přístupy k zabezpečení MQTT komunikace. Dají se rozdělit na tři základní typy, a to podle vrstev, kde zabezpečení probíhá [64]. **Síťová vrstva** může být zabezpečena pomocí VPN, kdy veškerá komunikace mezi klientem a serverem je šifrována, a tím ochráněna před zneužitím. Tato technologie je ale pro omezená zařízení příliš náročná. Do této kategorie by se dalo zařadit i použití *Virtual local area network* (VLAN) (nebo jiné fyzické oddělení), pro izolování MQTT provozu. Toto zabezpečení je sice možné použít, ale v případě bezdrátových technologií je stále možné provoz odposlouchávat i bez přímého fyzického přístupu k médiu. **Transportní vrstva** se nejčastěji zabezpečuje pomocí TLS, kdy celá MQTT zpráva je zapouzdřena v zašifrovaném TCP paketu a je tak ochráněna před odposlechem. Nevýhoda je velká výpočetní a paměťová náročnost tohoto protokolu, proto jej na mnoho omezených zařízení není možné implementovat. Na poslední **aplikační vrstvě** se například nachází dříve zmíněná nedostatečná MQTT autentizace. Řadí se sem také metoda **paylaod encryption**, kterou využívá navržený systém. Payload je část MQTT zprávy, která obsahuje přenášená data (např. aktuální teplota, koncentrace CO2, atd.). Tato metoda (paylaod encryption) šifruje pouze tuto část MQTT zprávy, zbytek zprávy zůstává beze změny. To má dvě výhody. Broker směruje zprávy podle Client ID a jiných parametrů a o payload zprávy se nezajímá. Celý proces šifrování a dešifrování je tedy pro broker neviditelný. Díky tomu, na rozdíl od jiných metod, není nutná náročná konfigurace brokeru. Druhou výhodou je, že se šifrují pouze nejnutnější data (nikoliv celá zpráva), a to snižuje výpočetní náročnost a režii systému. Nevýhodou této metody je problematická distribuce klíčů neboli jaký klíč má klient k šifrování použít.

6.1 Architektura systému

Každé MQTT téma v navrženém systému má vlastní kryptografický klíč (dále jen klíč tématu a `topic_key`). Pouze data zašifrovaná tímto klíčem jsou systémem přijata. V systému vystupuje *Key management system* (KMS), to je centrální autorita udržující databázi klíčů. Jakmile chce klient zaslat nějaká data, musí nejprve požádat KMS o klíč daného tématu, do které chce data posílat. KMS provede autentizaci a autorizaci klienta, následně klientovi přidělí klíč. Klient získaný klíč použije k šifrování dat. KMS také určuje, kolikrát se daný klíč může použít k šifrování. Po překročení limitu je klíč smazán a je nutné vyjednat nový. Zde nastává problém, jakým způsobem autentizovat klienta a jak mu dopravit klíč tématu, aniž by ho někdo odposlechl.

Autentizace klienta a ustanovení klíče

Schéma využívá asymetrickou kryptografii. Klient i KMS má veřejný a soukromý klíč. KMS uchovává databázi klientů (Client ID) a jejich veřejných klíčů. Klient zná veřejný klíč KMS a svůj soukromý klíč. Žádost o klíč tématu je šifrovaná dočasným klíčem relace (dále jen klíč relace a `temp_key`). Tento klíč je pro každou žádost unikátní. Vypočte se dle obrázku 6.2. Autentizaci klienta zaručuje Schnorrovo podpisové schéma. Ustanovení klíče Diffie–Hellman protokol.



Obr. 6.2: Kryptografické jádro systému

Schéma začíná vybráním náhodného čísla `r`, následně výpočtem závazku `C`. Hodnota `e` zde představuje výstup z hašovací funkce `H`, do které vstupují vstupují kromě závazku `C` také Client ID hodnota `nonce`. Nonce je náhodná a nepředvídatelná hodnota, kterou si klient vyžádal od KMS před začátkem výpočtu. To zamezuje mimo

jiné útokům opakováním žádostí. Client ID vstupuje do haše proto, aby se klient nemohl vydávat za jiného klienta. Následuje výpočet `z`. Podpis (`e, z`) a Client ID jsou následně předány KMS. KMS vypočte `A`, které se dle rovnice 6.1 rovná `C`.

$$A = pk^e \cdot g^z = g^{sk \cdot e} \cdot g^{r - e \cdot sk} = g^{e \cdot sk} \cdot g^{-e \cdot sk} \cdot g^r = g^r = C \tag{6.1}$$

KMS poté provede haš z Client ID, za které se klient vydává, vypočteného `A` a hodnoty nonce, kterou na začátku vyjednávání KMS zaslalo klientovi. Výsledný otisk se musí rovnat otisku, který klient zaslat v podpisu. Pokud se rovnají, autentizace byla úspěšná a může se pokračovat ve výpočtu klíče relace. Pokud se nerovnají, autentizace selhala a proces vyjednávání končí. Klíč relace se vypočte dle rovnice 6.2. Získaný klíč relace je nyní možné použít k zašifrování klíče tématu. Pouze ten klient, který byl ve vyjednávacím procesu autentizován, je schopen zprávu dešifrovat.

$$\texttt{temp\_key} = A^{SK_{KMS}} = (g^r)^{SK_{KMS}} = g^{r \cdot SK_{KMS}} = (g^{SK_{KMS}})^r = PK_{KMS}^r \tag{6.2}$$

Eliptické křivky a šifrování

Schéma popsané v předchozí části je založeno na problému diskrétního logaritmu *Discrete Logarithm Problem* (DLP) dle rovnice 6.3, kde x, y, g a p jsou přirozená čísla. Poté x je diskrétní logaritmus o základu g modulo p. Problém spočívá v tom, že ze znalosti g, x a p je velmi jednoduché spočítat y, ale ze znalosti g, y a p je velmi obtížné spočítat x. Tato vlastnost problém předurčuje k použití v asymetrické kryptografii, kde x obvykle představuje soukromý klíč a y klíč veřejný. Každé schéma založené na DLP je převoditelné na *Elliptic curve discrete logarithm problem* (ECDLP). Operace mocnění se zde mění na operaci násobení bodu na eliptické křivce skalárem podle rovnice 6.4. V této rovnici Y a G představují body na eliptické křivce a x je přirozené číslo. Je jednoduché získat bod Y vynásobením bodu G číslem x, ale je velmi složité ze znalosti Y a G získat x. Výhoda eliptických křivek je, že zaručují stejnou míru zabezpečení, za použití podstatně menšího klíče. Například 256 bitový ECC klíč zhruba odpovídá 3072 bitovému klasickému klíči. Z tohoto důvodu navržený systém implementuje kryptografické jádro nad strukturou eliptických křivek, to mu umožňuje značné výkonové a paměťové úspory. Systém využívá eliptickou křivku `secp256r1`. Tato křivka byla vybrána, protože poskytuje dostatečnou úroveň zabezpečení a podporují ji použité knihovny.

$$y \equiv g^x \ (mod\ p) \tag{6.3}$$

$$Y = x \cdot G \tag{6.4}$$

Veškeré šifrování v systému zajišťuje šifra AES v autentizovaném módu CCM o délce klíče 128 bitů. Tento mód de facto kombinuje *cipher block chaining message*

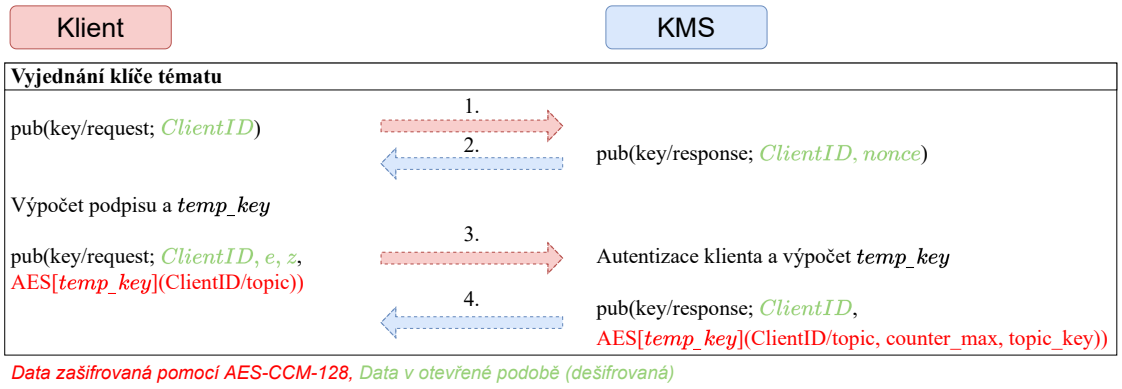
authentication code (CBC-MAC) s provozním režimem číslování. Do šifry vstupuje klíč, otevřený text, nonce a volitelně také *Additional authenticated data* (AAD), ty však v navrženém systémy použity nejsou. Výstupem šifry je šifrový text a autentizační tag. Nonce je náhodná hodnota, která musí být unikátní pro každé šifrování. Nonce, šifrový text a autentizační tag se následně přepošle druhé straně, která zprávu dešifruje a tím získá otevřený text a MAC (message authentication code), který porovná se zaslaným autentizačním tagem. Pokud se shodují, do zprávy nebylo nijak zasaženo. Tento mód tedy zajišťuje autentičnost, ale také integritu přenášených dat. Výhoda tohoto módu je, že prakticky funguje v proudovém módu, kdy se zvlášť vypočítává klíč, který se následně operací *Exclusive or* (XOR) bit po bitu „smíchává“ s otevřeným textem. Výsledný šifrový text je tak stejně dlouhý jako otevřený text (k šifrovému textu je pouze následně přidán autentizační tag). Neprovádí se zde žádná výplň. Podobným módem, co se vlastností týče, je *Galois/Counter Mode* (GCM), ten však nebylo možné použít, protože jej použité knihovny nepodporovaly.

Schéma komunikace

V systému jsou vyčleněny dvě MQTT témata pro vyjednávání o klíčích. Do tématu `key/request` zasílají klienti svoje žádosti a KMS jim do tématu `key/response` odpovídá. Na obrázku 6.3 je vidět celý proces ustanovení klíče tématu. `Pub` ve schématu značí, že se jedná o zprávu typu publish. Uvnitř závorky je název cílového tématu, do kterého je zpráva zaslána a za středníkem se nachází payload zprávy. Červeně je označen payload, který je zašifrovaný. Zelený je payload v otevřené podobě. V případě šifrového textu je v hranatých závorkách uveden klíč, kterým jsou data zašifrována. Ve schématu je vynechán broker. Veškerá MQTT komunikace nejdříve směřuje k brokeru a je následně přesměrována ke klientovi resp. KMS. Pro snazší pochopení je zde broker vynechán. Taktéž je zde vynecháno přihlášení se k tématům. Před započetím komunikace musí být KMS přihlášen k tématu `key/request` a klient k tématu `key/response`.

Vyjednávání začíná v kroku jedna, kdy klient zašle do tématu `key/request` zprávu obsahující svoje Client ID. Jakmile KMS obdrží zprávu obsahující pouze Client ID, tak vygeneruje náhodný nonce, který si uloží po dobu platnosti výzvy do databáze a v kroku dva jej společně s Client ID zašle klientovi. Klient následně vypočítá klíč relace a svůj podpis dle obrázku 6.2. Následně sestaví payload žádosti. Ten se skládá z Client ID, podpisu (e, z) a šifrového textu vzniklého zašifrováním názvu tématu, o který klient žádá (součást šifrového textu je i autentizační tag). K šifrování použije vypočtený klíč relace. Tuto zprávu odešle brokeru do tématu `key/request`. KMS z obdržené zprávy dekóduje Client ID a podpis, který dle ob-

rázku 6.2 ověří, a tím autentizuje klienta. Následně vypočte klíč relace, který použije k dešifrování zprávy. Po úspěšné autorizaci KMS vygeneruje nový klíč tématu, který si uloží do databáze. Poté klíčem relace zašifruje název tématu (o které klient žádá), hodnotu `counter_max` (kolikrát se může daný klíč použít k šifrování - tuto hodnotu určuje pro každé téma zvlášť uživatel, pokud ji neurčí, nastaví se výchozí hodnota) a samotný klíč tématu. KMS následně smaže nonce a klíč relace, poté si nastaví hodnotu `counter` v databázi na nula a odešle šifrový text spolu s Client ID v kroku čtyři do tématu `key/response`, kde naslouchá klient. Klient zprávu dešifruje dříve získaným klíčem. Pokud se téma ze zprávy rovná tématu o které klient žádal, tak smaže nonce a klíč relace, poté si klíč tématu spolu s hodnotu `counter_max` uloží do paměti, a také si v paměti nastaví hodnotu `counter` na nula, tato hodnota představuje aktuální číslo zprávy. Tím je celý proces vyjednávání klíče tématu hotov.

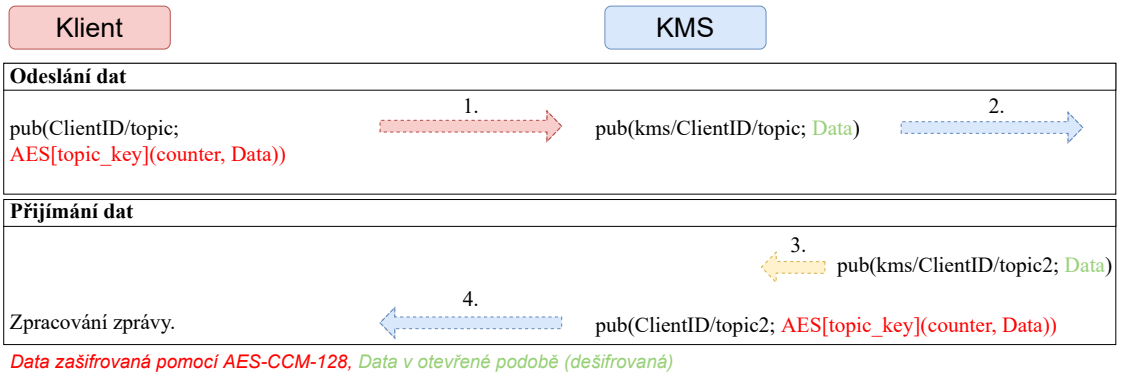


Obr. 6.3: Vyjednání klíče tématu

Zaslání dat do tématu je znázorněno na obrázku 6.4. Jakmile klient chce odeslat data, nejprve se ujistí, že k danému tématu má klíč (pokud ne, tak spustí vyjednávání), následně porovná hodnotu `counter` s hodnotou `counter_max`. Pokud je `counter` menší, tak se zvýší o jedna a přejde se k odeslání dat. Pokud se hodnoty rovnají, tak se jedná o poslední zprávu, která může být šifrovaná daným klíč tj. po odeslání zprávy klient smaže klíč a spustí vyjednávání. Klient nato aktuální hodnotu `counter` společně s daty zašifruje (klíčem tématu) a v kroku jedna jej odešle brokeru do příslušného tématu. KMS zprávu dešifruje (pokud má daný klíč) a porovná `counter` ze zprávy s hodnotou `counter` ve svojí databázi. Pokud je hodnota ze zprávy vyšší než hodnota v databázi, tak zprávu přijme a `counter` v databázi přepíše hodnotou `counter` ze zprávy. Pokud je `counter` ve zprávě menší nebo roven hodnotě v databázi, tak se jedná o útok opakováním, taková zpráva je zahozena. Pokud je hodnota ze zprávy rovna nebo vyšší než `counter_max`, tak se jedná o poslední zprávu do tématu, KMS zprávu přijme a následně smaže klíč tématu. KMS

poté získaná data ze zprávy vloží do nové zprávy, kterou v kroku dva odešle brokeru. Tato nová zpráva není zaslána do původního tématu, ale do nového. Toto téma vznikne spojením původního tématu s tématem `kms/` neboli původní téma se stane podtématem tématu `kms/`. Právo číst a zapisovat do podtémat `kms/` mají pouze důvěryhodná zařízení, těmi jsou zde myšleny KMS a služby interpretující data. Zajištění této podmínky bude rozebráno později.

Přijímání dat funguje stejným způsobem jako odesílání. Jakmile chce klient přijímat data, tak zjistí jestli má daný klíč tématu, pokud ne, tak si jej vyjedná viz obrázek 6.3, následně čeká na příchozí zprávy. Zaslání dat klientovi probíhá tak, že důvěryhodné zařízení zašle data podtématu `kms/`, kde naslouchá KMS. Toto je znázorněno v kroku tři na obrázku 6.4. Jakmile KMS obdrží tuto zprávu, tak odstraní z tématu řetězec `kms/`, tím získá téma, do kterého mají být data zaslána. Následně se podívá, jestli k danému tématu má v databázi klíč a data odešle. Odeslání dat funguje stejným způsobem, jako kdyby odesílal data klient viz odstavec výše. KMS tedy k datům přidá aktuální `counter`, poté jej zašifruje a v kroku čtyři odešle. Pokud KMS klíč nemá (téma neexistuje, nebyl vyjednán klíč, atd.), tak zprávu zahodí. Klient zprávu ověří stejným způsobem jako KMS v odstavci výše.



Obr. 6.4: Odesílání a příjímání dat

Výhodou tohoto řešení je, že interpretaci dat a zasílání dat klientům může provádět běžný MQTT klient, tj. neprovádí žádné šifrování, data přijímá i odesílá v otevřené podobě. O šifrování zpráv se stará KMS a klient. V navrženém systému si tedy uživatel může sám zvolit službu, která bude interpretovat naměřená data, respektive odesílat data klientům. Pouze je třeba zaručit, že daná služba smí číst a zapisovat do podtémat tématu `kms/`.

Důvěryhodná zařízení

V předchozí části bylo popsáno, jakým způsobem je zaručena integrita, důvěrnost a autentičnost přenášených zpráv. Tento mechanismus ale spoléhá na to, že právo číst a zapisovat do podtémat `kms/` smějí pouze důvěryhodná zařízení. Pokud by se útočník přihlásil k odběru podtémat `kms/`, tak by mu chodili veškeré zprávy v systému v otevřené podobě. Je tedy nutné kontrolovat přístup k těmto tématům. Důvěryhodné zařízení je z pohledu navrženého systému KMS a uživatelova služba nebo služby interpretující nebo zasílající data klientům. Tuto podmínku je nutné zajistit na brokeru, protože právě ten rozhoduje jaká zpráva má být komu přeposlána. Broker umožňuje kontrolovat přístup k tématům pomocí pravidel *Access-control list* (ACL). Tato přístupová pravidla je možné aplikovat buďto na konkrétní Client ID nebo na MQTT uživatele. Client ID se použít nedá, protože by si útočník mohl jednoduše své Client ID změnit a vydávat se tak za důvěryhodné zařízení. Je tedy nutné využít MQTT autentizaci, kdy důvěryhodné zařízení se nejprve autentizuje vůči brokeru, který mu následně povolí přístup do podtémat `kms/`. Zde nastává problém, viz sekce MQTT, že heslo klient posílá v otevřené podobě. Je tak nutné zajistit, že tuto zprávu nikdo nemůže odposlechnout. V sekci MQTT jsou také rozebrány možné přístupy jak toto zabezpečit. V případě, že se důvěryhodné zařízení nachází fyzicky na stejném zařízení jako broker, tak tento problém není třeba řešit, protože komunikace probíhá lokálně a nikdy neopustí zařízení. V případě, že broker a důvěryhodné zařízení jsou fyzicky na jiném místě, je nutné mezi nimi zajistit bezpečný kanál. Na to je možné využít například technologii VPN. V této konkrétní implementaci systému (viz Implementace) běží broker, KMS i interpretační služba, tedy všechna důvěryhodná zařízení na stejném serveru, a tak mezi nimi existuje bezpečný kanál a není tedy nutné implementovat další bezpečností mechanismy. Omezení přístupu k tématu `kms/` tedy funguje tak, že broker umožní čtení a zápis pouze autentizovaným zařízením. Autentizace probíhá standardním způsobem, ale je nutné zajistit bezpečný kanál mezi autentizovaným zařízením a brokerem. Existují i jiné metody jak omezit přístup k tématům například omezení podle příchozího rozhraní a MAC adresy, nebo zavedení dvou brokerů, kdy jeden komunikuje pouze s důvěryhodnými zařízeními a druhý pouze s klienty a KMS. Použité řešení je za splnění podmínky (bezpečný kanál mezi brokerem a důvěryhodným zařízením) jednoduché a funkční.

Bezpečnost

Navržený systém zaručuje autentičnost, důvěrnost a integritu přenášených dat. Každý klient v systému má svůj veřejný a soukromý klíč. Jakmile chce klient přijímat nebo zasílat data do tématu, tak spustí vyjednávání o klíč tématu. Žádost o klíč tématu je šifrována klíčem relace. Klíč relace je pro každé vyjednávání unikátní,

protože je vypočten z výzvy od KMS. KMS na základě žádosti klienta autentizuje a autorizuje a následně vygeneruje náhodný klíč tématu, který zašle klientovi. Každý klíč tématu je omezen počtem použití. Systémem jsou přijaty pouze zprávy, které mají vyšší hodnotu `counter` než zpráva předchozí. To zaručuje ochranu proti útokům opakováním. Tento útok nelze aplikovat ani během vyjednávání, protože výzva od KMS je platná pouze na určitou dobu a je přijata pouze jednou (KMS výzvu i klíč relace po úspěšném vyjednání smaže). Ze stejného důvodu nebude úspěšný ani útok na klienta. Zprávy jsou šifrovány šifrou AES-CCM, to zajišťuje, že během přenosu nemůže dojít k úpravě zprávy. V případě kompromitace klíče tématu může KMS kdykoliv klíč odvolat (smaže jej). Poté bude již následující zpráva odmítnuta (nebude ji možné dešifrovat). V případě kompromitace celého klienta KMS smaže klientův veřejný klíč. Do jednoho tématu (netýká se `key/request`) smí zapisovat pouze jeden klient. Klient nemůže zapisovat do cizích témat, protože smí zapisovat pouze do podtémat tématu svého client ID. Toto pravidlo nezajišťuje broker, ale KMS. KMS zašle klíč tématu pouze autentizovanému klientovi, který žádá o téma na které má právo. KMS přepošle do podtémat `kms/` pouze zprávy, které se mu podaří dešifrovat (aby se mu to podařilo, zpráva musí pocházet od autentizovaného a autorizovaného klienta, protože pouze tomu KMS zašle klíč tématu).

6.2 Implementace

Hardware

V implementovaném systému v roli klientů vystupují zařízení ESP32 a ESP8266. Tato zařízení byla vybrána z důvodu jejich relativně vysokého výkonu a také proto, že se během výkonnostních testů ukázali jako nejméně problémové. Tyto desky také obsahují WiFi modul, který je potřebný ke správné funkci systému. Systém je možné implementovat i na jiných deskách kompatibilních s RIOT OS, WiFi a použitými knihovnami. Systém nijak nespecifikuje data, které jím mohou být zasílána. V tomto případě byli někteří klienti doplněni o senzor BME280 [65] v zapojení podle tabulky 6.1. Tento senzor byl následně použit k měření teploty a vlhkosti v sledovaných místnostech, takto získaná data byla následně zasílána serveru. Úloha serveru je v systému zajištěna pomocí mikropočítače Raspberry Pi 4 [66].

Software

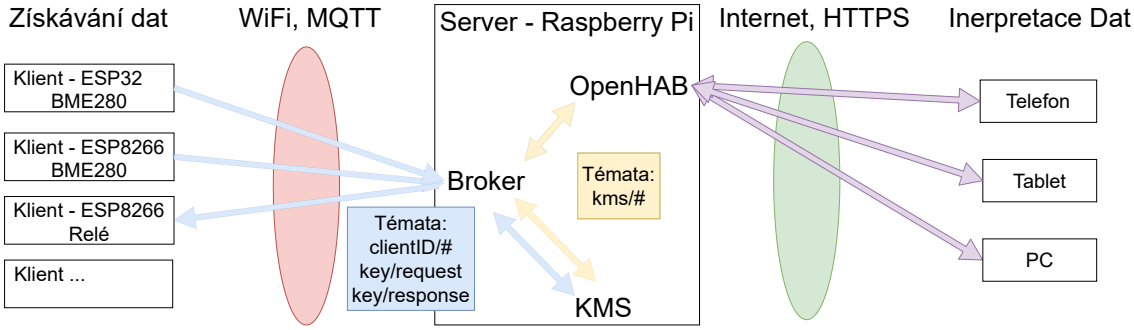
Aplikace běžící na klientech jsou RIOT OS kompatibilní. Tento operační systém byl popsán v kapitole 2. Vývoj aplikací probíhal stejným způsobem, který byl popsán v kapitole 4. MQTT komunikaci zajišťuje knihovna Paho MQTT [67]. Operace nad strukturou eliptických křivek provádí knihovna Micro-ECC [68]. Tato knihovna byla

Tab. 6.1: Zapojení senzoru BME280

BME280	ESP32	ESP8266
VIN	3V3	3V3
GND	GND	GND
SCL	G22	D1
SDA	G21	D2

i přes horší výsledek upřednostněna před knihovnou Relic z důvodu menší paměťové náročnosti. Šifra AES-CCM je v systému implementována pomocí RIOT modulu Crypto [69]. Na serveru běží open-source MQTT broker Mosquitto [70], a také KMS, které je implementováno pomocí skriptu v jazyce Python. KMS využívá taktéž knihovnu Paho MQTT. Veškeré kryptografické operace zajišťuje knihovna PyCryptodome [71]. Systém byl doplněn o službu OpenHAB [72]. OpenHAB je open-source automatizační nástroj, který dokáže mimo jiné přijímat a odesílat MQTT zprávy do navrženého systému (podtémata `kms/`). Takto získaná data následně umí ukládat a interpretovat uživateli. KMS a OpenHAB jsou z pohledu systému důvěryhodná zařízení viz sekce 6.1. Na brokeru je tedy nutné zajistit dva uživatelské účty, vůči kterým se musí autentizovat. V případě, že by se nacházeli na fyzicky jiném zařízení než broker, je nutné zajistit šifrování provozu.

Schéma implementovaného systému je vidět na obrázku 6.5. Obrázek je rozdělen do třech částí. V levé části je IoT síť tvořena jednotlivými klienty. Uprostřed je server a služby na něm běžící. V pravé části jsou zařízení, kterým jsou interpretována data pocházející ze systému. Červená elipsa na obrázku reprezentuje technologii WiFi, pomocí které se klienti připojují k serveru. Zelená elipsa představuje internet nebo jinou síť, z které se bezpečným způsobem připojují zařízení, která zobrazují data ze systému a zasílají příkazy klientům. Tato zařízení nejsou součástí systému, pouze interagují se systému pomocí služby OpenHAB, která je z pohledu systému důvěryhodné zařízením.



Obr. 6.5: Schéma implementovaného systému

Modré šipky představují MQTT komunikaci, která směřuje do/z témat vypsaných v modré tabulce. Žluté šipky zastupují MQTT komunikaci směřující do/z podtémat, do kterých smí zapisovat/číst pouze důvěryhodná zařízení (žlutá tabulka). Fialové šipky reprezentují *Hypertext Transfer Protocol Secure* (HTTPS) komunikaci mezi službou OpenHAB jejími klienty.

Konfigurace brokeru

Většinu úloh v systému provádí KMS, proto broker vyžaduje pouze minimální konfiguraci. Konfigurační soubor brokeru je vidět na výpisu 6.1. Zde je třeba povolit anonymní přihlašování (klienti MQTT autentizaci nevyužívají). Následně je nutné specifikovat soubor obsahující jména a hesla důvěryhodných zařízení oddělená dvojtečkou. Tento soubor se zašifruje pomocí příkazu `mosquitto_passwd -U pass`. Poslední řádek přidává soubor s ACL pravidly. Tento soubor (acl.acl) je vidět na výpisu 6.2. První pravidlo, platící pro všechny anonymní uživatele (klienty), povoluje čtení a zápis pouze do podtémat tématu svého client ID. `%c` zde představuje client ID příchozí zprávy a `#` je zástupný symbol pro libovolné množství podtémat. Další dvě pravidla rozšiřují práva klientů o zápis, respektive čtení témat nutných k vyjednávání klíče. Zbylá pravidla opravňují důvěryhodná zařízení k zápisu/čtení do všech témat.

Výpis 6.1: Konfigurační soubor brokeru

```
listener 1883                                    1
protocol mqtt                                    2
allow_anonymous true                             3
password_file /etc/mosquitto/pass                4
acl_file /etc/mosquitto/acl.acl                  5
```

Výpis 6.2: Soubor obsahující ACL pravidla brokeru

```
pattern readwrite %c/#                           1
topic write key/request                          2
topic read key/response                          3
user kms                                         4
topic readwrite #                                5
user openhab                                     6
topic readwrite #                                7
```

Klientská aplikace

Systém je pro klienty dostupný ve formě RIOT modulu, stačí jej tedy připojit k programu a následně využívat jeho funkce. Část souboru Makefile, zajišťující připojení

navrženého systému do aplikace klienta, je vidět na výpisu 6.3. V tomto souboru se kromě připojení modulu také nastavuje přihlašování k WiFi, client ID klienta a IP adresa brokeru. Následně je možné pomocí direktivy `#include` importovat systém a přistupovat k jeho funkcím.

Výpis 6.3: Část souboru Makefile připojující navržený systém

```
...
# Nastavení
WIFI_SSID ?= "wifi"
WIFI_PASS ?= "heslo"
MQTT_CLIENT_ID ?= "clientID1"
MQTT_BROKER_IP ?= "192.168.0.1"
# Absolutní cestu do adresáře systému
INCLUDES += -I$(CURDIR)/../../mqtt-security
DIRS += $(CURDIR)/../../mqtt-security
USEMODULE += mqtt-security
...
```

Příklad, jak může vypadat konkrétní aplikace klienta využívající navržený systém, je vidět na výpisu 6.4. V aplikaci je z důvodu úspory místa vynechán klíč klienta a KMS. Pro úspěšnou kompilaci je nutné je přidat. Tento program se pomocí systému přihlásí k tématu `prepinac` a každou půl minutu odesílá řetězec `Data` do tématu `teplota`. Jakmile uživatel (prostřednictvím OpenHAB) zašle data klientovi do tématu `prepinac`, tak se spustí funkce `function`, která data vypíše. Výstup z konzole klienta je vidět na obrázku 6.6. Na obrázku 6.7 je vidět výpis z logovacího souboru KMS. Zde je vidět veškerá komunikace mezi klientem a KMS. Červenou čarou je pro lepší přehlednost odděleno vyjednávání o klíč tématu (první čtyři zprávy), odeslání dat do tématu `teplota` a přijmutí dat z tématu `prepinac`. Vyjednávání klíče tématu probíhá dle schématu komunikace 6.3, které bylo popsáno výše. První zachycená zpráva byla zaslána do tématu `key/request` a obsahuje pouze client ID klienta 3. Jedná se tedy o žádost klienta o `nonce`. Druhá zpráva byla zaslána do tématu `key/response` a pochází tedy od KMS. KMS zde odpovídá klientovi zasláním hodnoty `nonce`. Následuje žádost klienta o klíč tématu zašifrovaná klíčem relace. Čtvrtá zpráva je odpověď od KMS obsahující klíč tématu zašifrovaný klíčem relace. Tyto čtyři zprávy představují vyjednávání klíče tématu. V tomto případě má klient dvě témata (`prepinac, teplota`), a tak musely proběhnou dvě vyjednávání klíče tématu. Další zpráva v pořadí přišla do tématu `clientID3/teplota`. Klient zde zasílá data do tématu `teplota` zašifrovaná klíčem tématu. Zpráva je úspěšně autentizována a v další zprávě je přeposlána do podtématu `kms`, kde si jej přebírá důvěryhodné zařízení. Poslední zvýrazněná dvojice zpráv představuje komunikaci opačným směrem, tj. důvěryhodné zařízení → klient.

Výpis 6.4: Aplikace klienta využívající navržený systém

```
#include "mqtt_security.h"
static MQTTClient client;
MQTTTopic teplota;
MQTTTopic prepinac;
uint8_t KMSPublic[KEY_SIZE * 2] = {0x99, 0x11, 0x89, ...};
uint8_t ClientSecret[KEY_SIZE] = {0x1D, 0x2F, 0xBF, ...};
void function(uint8_t * data, unsigned int data_size) {
  data[data_size] = '\0';
  printf("Data: %s\n", (char *)data);
  return;
}
int main(void) {
  mqtt_connect(&client);
  setKeys(ClientSecret, KMSPublic);
  createTopic(&teplota, "teplota");
  createTopic(&prepinac, "prepinac");
  receiveEncryptedData(&prepinac, function);
  char data[] = "Data";
  while(true){
    sendEncryptedData(&teplota, &data, strlen(data));
    xtimer_sleep(30);
  }
  return 0;
}
```

```
2021-05-09 14:58:07,873 # main(): This is RIOT! (Version: 2020.07)
2021-05-09 14:58:10,791 # WiFi connected to ssid Opice, channel 13
2021-05-09 14:58:29,554 # Data: test
2021-05-09 14:58:32,958 # Data: test2
```

Obr. 6.6: Výpis z konzole klienta

```
pi@raspberrypi:~/KMS $ tail log -n 14
14:58:16 topic: key/request payload: clientID3
14:58:16 topic: key/response payload: clientID3H]. P'
14:58:17 topic: key/request payload: clientID3C<CM?M:Z<7U_6z J|Y U
f u D X , k ba\
14:58:17 topic: key/response payload: clientID3H]. P' u D X , m fp, 1s o ^ t  d
14:58:18 topic: clientID3/teplota payload: h ^ gZ W ^v r 1 w
14:58:18 topic: kms/clientID3/teplota payload: Data
14:58:29 topic: kms/clientID3/prepinac payload: test
14:58:29 topic: clientID3/prepinac payload:  j2q t  /?<$
14:58:32 topic: kms/clientID3/prepinac payload: test2
14:58:32 topic: clientID3/prepinac payload: u(C M Jq dk.cyT 1
14:58:48 topic: clientID3/teplota payload: m fS Kj Ip f O
14:58:48 topic: kms/clientID3/teplota payload: Data
14:59:18 topic: clientID3/teplota payload: 8  _        @    7y$C k
14:59:18 topic: kms/clientID3/teplota payload: Data
```

Obr. 6.7: Výpis z logovacího souboru KMS

Všechny funkce systému dostupné uživateli jsou popsány v hlavičkovém souboru `mqtt-security.h`, proto zde nebudou jednotlivě rozebírány. V tomto souboru se také nachází řada nastavení, kterými může uživatel ovlivnit chování systému. Jedná se například o zapnutí výpisu do konzole, automatické přidávání client ID před uživatelova témata, časovač, jak dlouho může klient čekat na odpovědi od KMS, a tak dále.

KMS

KMS je v systému implementována pomocí Python 3 skriptu. Ten vyžaduje knihovnu `paho-mqtt`. Dále knihovnu `pycryptodome`, která zajišťuje šifrování, hašování a operace s body na eliptické křivce. Další potřebnou knihovnou je `json`, zajišťující načtení dat z databáze. Poslední knihovnou je `datetime` umožňující práci s časem (pro logování a kontrolu platnosti výzvy). Aby bylo možné KMS spustit, tak musí ve stejném adresáři existovat soubor `database.txt`. V tomto souboru se nachází konfigurace a databáze KMS. Příklad, jak by takový soubor mohl vypadat, je na výpisu 6.5. Soubor je ve formátu JSON. Klíče klientů/KMS zde nejsou vypsány celé. Soubor je rozdělen do tří části. V poslední sekci `clients` je databáze klientů. Každý klient v systému musí mít unikátní client ID a veřejný klíč. Ten je formátu jako u knihovny Micro-ECC, tedy x a y souřadnice EC bodu za sebou. Jednotlivé bajty jsou zapsány hexadecimálně bez mezer. Klient, který není v databázi nebo je v databázi, ale nemá veřejný klíč, tak bude systémem odmítnut. V sekci `topics` je databáze témat. V této databázi je možné specifikovat hodnotu `coutner_max`, která určuje kolikrát se může šifrovat jedním klíčem. V sekci `settings` je nastavení KMS včetně jeho privátního klíče. Zde je opět možné ovlivňovat chování systému. Důležitá položka je například `allow_unknown_topics`. Pokud je nastavena na `false`, tak je možné vyjednat klíč tématu pouze k tématům, která jsou explicitně vypsána v sekci `topics`. Pokud je hodnota `true`, tak je možné vyjednat klíč k tématu, které není v databázi. V takovém případě bude nastavena defaultní hodnota `coutner_max`. Dále například položka `logging`, která zapne logování veškerých zpráv v systému.

Generování klíčů

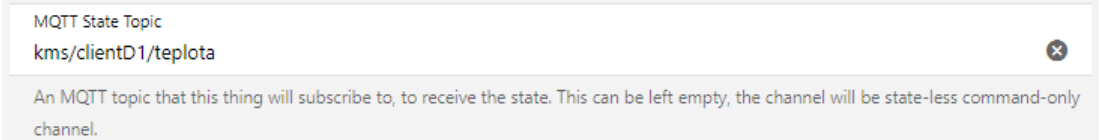
Každý klient a KMS musí mít soukromý a veřejný klíč. Soukromý klíč představuje číslo, jejímž vynásobením vznikne klíč veřejný. Respektive vynásobením generátoru křivky soukromým klíčem vznikne bod, který představuje klíč veřejný. Generování klíčů je tedy poměrně jednoduché, stačí vygenerovat náhodné číslo a vynásobit jím generátor křivky. V příloze práce je python skript, který právě toto provádí a zároveň vypíše klíče ve správném formátu pro KMS i klienta, které je následně možné vložit do kódu klienta, respektive nastavení KMS.

Výpis 6.5: Konfigurační/databázový soubor KMS

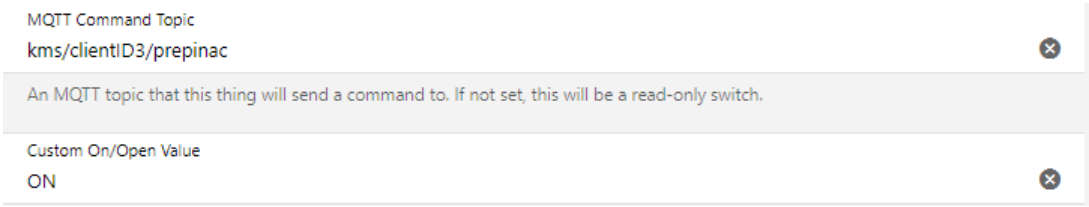
```
{"settings":{
    "kms_clientID": "kms",
    "kms_username": "kms",
    "kms_password": "heslo",
    "broker_address": "192.168.0.1",
    "nonce_validity_time": 10,
    "allow_unknown_topics": true,
    "counter_max_default": 255,
    "show_debug_info": true,
    "show_sensitive_info": false,
    "logging": true,
    "private_key": "C1DE7F27F6B6F577511B27D37B2C04F0..."},
 "topics":{
    "clientID1/teplota": {"counter_max": 1111},
    "clientID1/vlhkost": {"counter_max": 64},
    "clientID3/prepinac": {"counter_max": 10}},
 "clients":{
    "clientID1": {"public_key": "A4DDFC0C72A44D4EC02..."},
    "clientID2": {"public_key": "DB3DC5734F8EA06B293..."},
    "clientID3": {"public_key": "09D080844DD73129F6..."}}}
```

6.3 Interpretace dat

Interpretaci dat může zajišťovat jakákoliv služba, která umí zpracovávat MQTT zprávy a splňuje podmínku důvěryhodného zařízení. V implementovaném systému byla zvolena služba OpenHAB, které výše uvedené vlastnosti splňuje. OpenHAB byl zvolen, protože umožňuje jednoduše vytvářet uživatelská rozhraní, která jsou následně dostupná z internetu. Službu interpretující data volí uživatel a není tak součástí navrženého systému, proto zde ani nebude do hloubky popisována konfigurace služby OpenHAB. Ve zkratce, probíhá tak, že se nejdříve přidá MQTT broker (pomocí lokální IP adresy 127.0.0.1, portu a přihlašovacích údajů). Následně je možné vytvářet tzv. `channels`, které se mapují na MQTT témata viz obrázek 6.8 a 6.9. S takto získanými daty je možné dále pracovat.



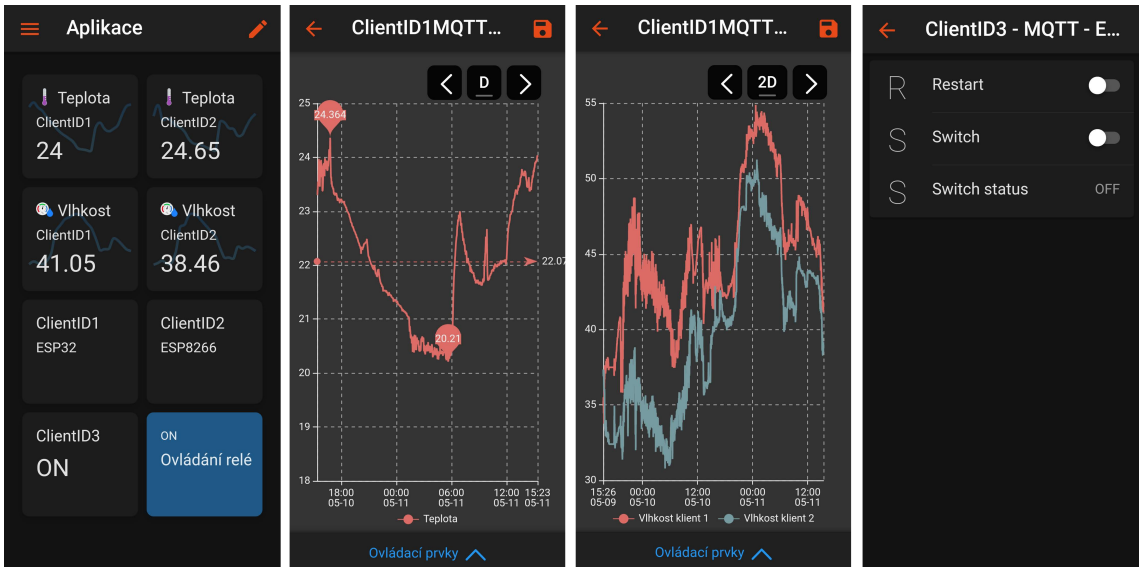
Obr. 6.8: OpenHAB konfigurace tématu pro přijímaní dat



Obr. 6.9: OpenHAB konfigurace tématu pro odesílání dat

Uživatelské rozhraní

Na obrázku 6.10 jsou vidět snímky obrazovky z mobilní aplikace. První snímek zleva je úvodní obrazovka. V horní části se zobrazuje aktuální teplota a vlhkost naměřená klienty jedna a dva. Jakmile uživatel na toto políčko klikne, tak se zobrazí graf. V grafu je možné měnit délku sledovaného období, zobrazovat minima a maxima nebo průměr. Grafy je také možné vzájemně kombinovat. To je vidět na třetím snímku, kde se porovnává vlhkost naměřená klientem jedna a dva. Poslední dvě políčka na úvodní obrazovce se týkají klienta tři, který neměří teplotu, ale ovládá relé. Po kliknutí na tlačítko Ovládání relé OpenHAB zašle příkaz klientovi, který následně sepne relé. Jakmile to provede, tak zašle uživateli potvrzení s aktuálním stavem. Klient tři takto využívá dvě témata. Na prvním tématu `clientID3/prepinac` naslouchá příkazy (které zašle OpenHAB po zmáčknutí tlačítka uživatelem) a do druhého tématu `clientID3/status` zasílá aktuální stav relé. Tento stav se následně uživateli vypíše na úvodní obrazovce v políčku ClientID3. Poslední snímek obrazovky je detail klienta tři. Detail klienta se zobrazí po kliknutí na client ID na úvodní obrazovce a zobrazuje všechny témata spjatá s daným klientem. Kromě dvou zmíněných témat má ještě každý klient téma `restart`, které umožňuje vzdáleně restartovat klienta. Stejné rozhraní je dostupně pomocí webového prohlížeče.



Obr. 6.10: Snímky obrazovky z mobilní aplikace

6.4 Výkonnostní test

Systém využívá několik kryptografických primitiv a algoritmů, viz sekce 6.1 Architektura systém. Klient konkrétně využívá modulární aritmetiku, hašování funkcí SHA-256, šifrování/dešifrování šifrou AES-CCM-128 a násobení bodu na eliptické křivce skalárem. Server používá vše zmíněné a navíc sčítá body na eliptické křivce. Výkonnostní testy jednotlivých operací byly provedeny v kapitole 5. Tato sekce se tak věnuje výkonu systému jako celku. Byly provedeny celkem tři testy. První měří jak dlouho klientovi trvá vytvořit žádost o klíč tématu. Tj. rozdíl času mezi obdržením `nonce` po odeslání žádosti. Jedná se tedy o výpočet klíče relace a sestavení žádosti. Používá se zde modulární aritmetika, hašovaní, šifrování a dvakrát násobení skalárem. Druhý test měřil jak dlouho klientovi trvá zpracovat odpověď od KMS. Konkrétně čas, kdy klient zjistí, že se jedná o odpověď od KMS po uložení klíče tématu. Zde se provádí pouze dešifrování, ale k tomu různé ověřování. Poslední test měřil čas celého vyjednávání, tj. od zaslání výzvy klienta po uložení klíče tématu. Jsou zde zahrnuty všechny kryptografické operace, jak na straně klienta, tak KMS. Výsledný čas tak může být ovlivněn řadou faktorů. Výsledek všech testů je vidět v tabulce 6.2. Výsledné časy jsou lepší než se očekávalo. Výpočet klíče relace trvá zhruba stejně dlouho jako dvojnásobné násobení skalárem, viz kapitola 5. Z toho vyplývá, že ostatní operace (hašovaní, šifrování, atd.) a režie samotného systému jsou v porovnání se skalárním násobením zanedbatelné. Zpracování odpovědi trvalo oběma deskám srovnatelnou dobu, přestože deska ESP32 má mnohem vyšší výkon. V tomto testu je také zahrnuto zrušení odběru k tématu `key/response`, kdy klient čeká na potvrzení od brokeru. To nejspíše tvoří značnou část naměřeného času. Poslední test měřil celé vyjednávání. Zde se očekávalo, že režie systému (zasílání MQTT zpráv, čekání na odpovědi, výpočty KMS, ...) bude mít zásadní dopad na výsledný čas. Po odečtení dvou předchozích měření od výsledného času se ukázalo, že režie systému tvořila pro desku ESP8266 **239** milisekund a pro desku ESP32 **178** milisekund. Tyto časy by jistě šly částečně zkrátit optimalizací kódu. Nicméně v porovnání se skalárním násobením již teď tvoří malou část výsledného času a dají se tak považovat za dostačující.

Tab. 6.2: Výkonnostní test implementovaného systému

	Výpočet klíče relace	**Zpracování odpovědi**	**Celé vyjednávání**
ESP8266	892 ms	35 ms	1131 ms
ESP32	535 ms	44 ms	757 ms

Závěr

Cílem bakalářské práce bylo nastudovat vývoj RIOT aplikací a analyzovat RIOT kompatibilní zařízení a kryptografické knihovny. Následně provést benchmarkové testy kryptografických knihoven na výkonově omezených zařízeních. Dále měla být provedena analýza možných komunikačních bezdrátových technologií, použitelných k bezpečnému sběru dat. Na základě nabytých znalostí měl být navržen a implementován systém umožňující sběr dat ze senzoru, který bude zajišťuje důvěrnost, autentičnost a integritu přenášených dat.

Práce se ve své teoretické části zabývala základy kryptografie, vysvětlila pojmy, se kterými dále pracovala. Byla zde popsána struktura RIOT, podporované architektury a kryptografické knihovny. Dále zde byl teoreticky popsán postup vývoje RIOT aplikací. Následující část analyzovala současné bezdrátové technologie, které by se daly použít k bezpečnému sběru dat v IoT síti.

V praktické části byl vytvořen návod ke zprovoznění vývojového prostředí, včetně konkrétních příkazů nutných ke kompilaci a nahrání kódu na zařízení. Byly zde popsány i různé problémy, které během tohoto procesu mohou nastat a postupy jakým způsobem je řešit. Následující kapitola se věnovala samotným výkonnostním testům. V první části je popis metod měření a seznámení s deskami, které byly k testům použity. Další sekce obsahují výsledky měření. Během testů byla objevena řada problémů, znemožňující měření na některých deskách. Zařízení Arduino Nano bylo obzvláště problematické, proto bylo z některých testů vyřazeno. Z výsledků vyplývá, že deska STM32F103C8T6 dosahuje výborného poměru cena/výkon, kdy se vyrovnala a často i překonala mnohem výkonnější i dražší zařízení ESP32. Tyto vlastnosti jsou však vykoupeny horší uživatelskou přívětivostí.

Poslední kapitola se věnuje návrhu a implementaci systému zajišťující bezpečný přenos libovolných dat mezi klientem a serverem. V úvodní části je popsán komunikační protokol MQTT, který byl pro svou jednoduchost a nenáročnost použit v systému. Jsou zde také popsány jednotlivé přístupy k zabezpečení komunikace a možné útoky. Na tomto základu je představen navržený systém a jednotlivé entity, které v něm vystupují. Poté je popsáno kryptografické jádro systému, použité šifry a důvod proč systém využívá eliptické křivky. Následuje popis jednotlivých zpráv, které systém využívá k vyjednávání klíčů a zasílání dat. Sekce 6.2, Implementace, líčí konkrétní knihovny a zařízení, které byly použity k implementaci systému. Je zde ukázáno, jakým způsobem přidávat klienty do systému, jak konfigurovat KMS a jakým způsobem generovat klíče. Další sekce se věnuje interpretaci naměřených dat. Je zde krátký popis služby, která byla k tomuto účelu použita. Nechybí ukázka uživatelského rozhraní, které ze/do systému data přijímá a odesílá. V poslední části kapitoly byl proveden výkonnostní test implementovaného systému.

K práci je přiložen ZIP archiv obsahující všechny získané výsledky. Archiv také obsahuje zdrojové kódy jednotlivých výkonnostních testů a kód implementovaného systému, včetně konfiguračních souborů. Struktura archivu je popsána v příloze A.

V souboru `benchmark.xlsx` jsou výsledky všech provedených výkonnostních testů. Některé testy se již do této práce nevešly a jsou tak uvedeny pouze v tomto souboru. Zdrojové kódy většiny testů jsou ve složce `benchmark`. V souboru `RIOT_manual.pdf` se nachází detailní návod na zprovoznění všech desek, které byly v práci použity. Ve složce `mqtt-security` jsou zdrojové kódy a konfigurační soubory implementovaného systému. Uvnitř této složky je python skript `keyGen.py` generující klíče ve správném formátu pro klienta i KMS. Je možné vygenerovat náhodný pár nebo pouze veřejný klíč/bod ze zadaného skaláru. V podsložce `KMS` je kód KMS a příklad konfiguračního a databázového souboru. V podsložce `broker` je konfigurační soubor brokeru a soubor s pravidly ACL. Ke zprovoznění brokeru je ještě nutné přidat soubor s přihlašovacími údaji důvěryhodných zařízení, viz sekce 6.2. V podsložce klienti jsou zdrojové kódy klientů, které byly použity v implementovaném systému. Před kompilací kódu je nutné se v souboru `Makefile` ujistit, že jsou správně nastaveny cesty do adresáře RIOT a samotného systému, který se nachází v podsložce `mqtt-security`.

Literatura

[1] *MENEZES, Alfred J., Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, 1997. ISBN 0-8493-8523-7.*

[2] *ROSER, Max a Hannah RITCHIE. Technological Progress. OurWorldInData.org [online]. 2013 [cit. 2020-11-24]. Dostupné z: <https://ourworldindata.org/technological-progress>*

[3] *RFC 7228 - Terminology for Constrained-Node Networks. 1. Bremen: Internet Engineering Task Force (IETF), 2014. Dostupné také z: <https://tools.ietf.org/html/rfc7228>*

[4] *Internet of Things Statistics, Facts & Predictions [2020's Update] [online]. 2020 [cit. 2020-12-10]. Dostupné z: <https://review42.com/internet-of-things-stats/>*

[5] *IoT botnets might be the cybersecurity industry's next big worry [online]. [cit. 2020-12-10]. Dostupné z: <https://www.iotsecurityfoundation.org/iot-botnets-might-be-the-cybersecurity-industrys-next-big-worry/>*

[6] *DATA ENCRYPTION STANDARD (DES). FIPS PUB 46-3. Gaithersburg: National Institute of Standards and Technology (NIST), 1999. Dostupné také z: <https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf>*

[7] *MCNETT, David. US GOVERNMENT'S ENCRYPTION STANDARD BROKEN IN LESS THAN A DAY [online]. DES Challenge III, 1999 [cit. 2020-11-07]. Dostupné z: <http://www.distributed.net/images/d/d7/19990119_-_PR_-_release-des3.pdf>*

[8] *Transitioning the Use of Cryptographic Algorithms and Key Lengths. Special Publication 800-131A. Gaithersburg: National Institute of Standards and Technology (NIST), 2018. Dostupné také z: <https://csrc.nist.gov/CSRC/media/Publications/sp/800-131a/rev-2/draft/documents/sp800-131Ar2-draft.pdf>*

[9] *Recommendation for Key Management: Part 3: Application-Specific Key Management Guidance. Special Publication 800-57. Gaithersburg: National Institute of Standards and Technology (NIST), 2015. Dostupné také z: <https:*

//nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf>

[10] *DAEMEN, Joan a Vincent RIJMEN. AES Proposal: Rijndael. Computer Security Resource Center (NIST) [online]. 3. 9.1999 [cit. 2020-11-07]. Dostupné z: <https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf>*

[11] *Announcing the ADVANCED ENCRYPTION STANDARD (AES). FIPS PUB 197. Gaithersburg: National Institute of Standards and Technology (NIST), 2001. Dostupné také z: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>*

[12] *RIVEST, Ronald L., Adi SHAMIR a Leonard M. ADLEMAN. Cryptographic communications system and method. US4405829A. Uděleno 20.9.1983. Dostupné také z: <https://patents.google.com/patent/US4405829A/en>*

[13] *LAKE, Josh. What is RSA encryption and how does it work? Comparitech [online]. 2018 [cit. 2020-11-07]. Dostupné z: <https://www.comparitech.com/blog/information-security/rsa-encryption/>*

[14] *HELLMAN, Martin E., Bailey W. DIFFIE a Ralph C. MERKLE. Cryptographic apparatus and method. US4200770A. Uděleno 29. 4. 1980. Dostupné také z: <https://patents.google.com/patent/US4200770>*

[15] *Digital Signature Standard (DSS). FIPS PUB 186-4. Gaithersburg: National Institute of Standards and Technology (NIST), 2013. Dostupné také z: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>*

[16] *Digital Signature Algorithm. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-11-07]. Dostupné z: <https://cs.wikipedia.org/wiki/Digital_Signature_Algorithm>*

[17] *RFC 1321 - MD5 Message-Digest Algorithm. 1. Cambridge: Network Working Group, 1992. Dostupné také z: <https://tools.ietf.org/html/rfc1321>*

[18] *MD5 vulnerable to collision attacks. CERT Coordination Center [online]. Carnegie Mellon University: Software Engineering Institute, 2008 [cit. 2020-11-07]. Dostupné z: <https://www.kb.cert.org/vuls/id/836068>*

[19] *Secure Hash Standard. FIPS PUB 180-2. Gaithersburg: National Institute of Standards and Technology (NIST), 2002. Dostupné také z: <*https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf*>*

[20] *PETERKA, Jiří. Datové schránky přešly na SHA-2. Lupa.cz [online]. 2010 [cit. 2020-11-07]. Dostupné z: <*https://www.lupa.cz/clanky/datove-schranky-presly-na-sha-2/*>*

[21] *Secure Hash Standard (SHS). FIPS PUB 180-4. Gaithersburg: National Institute of Standards and Technology (NIST), 2015. Dostupné také z: <*https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf*>*

[22] *RIOT OS [online]. Berlin, 2008 [cit. 2020-11-07]. Dostupné z: <*https://www.riot-os.org/*>*

[23] *RIOT/cpu/ [online]. [cit. 2020-11-07]. Dostupné z: <*https://github.com/RIOT-OS/RIOT/tree/master/cpu*>*

[24] *RIOT/boards/ [online]. [cit. 2020-11-07]. Dostupné z: <*https://github.com/RIOT-OS/RIOT/tree/master/boards*>*

[25] *Family: native [online]. [cit. 2020-11-13]. Dostupné z: <*https://github.com/RIOT-OS/RIOT/wiki/Family%3A-native*>*

[26] *RIOT OS wiki [online]. [cit. 2020-11-13]. Dostupné z: <*https://github.com/RIOT-OS/RIOT/wiki*>*

[27] *Packages. RIOT Documentation [online]. 2020 [cit. 2020-11-13]. Dostupné z: <*https://doc.riot-os.org/group__pkg.html*>*

[28] *Kmackay / micro-ecc [online]. [cit. 2020-11-13]. Dostupné z: <*https://github.com/kmackay/micro-ecc*>*

[29] *Relic-toolkit / relic [online]. [cit. 2020-11-13]. Dostupné z: <*https://github.com/relic-toolkit/relic*>*

[30] *TinyCrypt Cryptographic Library [online]. 2017 [cit. 2020-11-13]. Dostupné z: <*https://github.com/intel/tinycrypt/blob/master/documentation/tinycrypt.rst*>*

[31] *Intel / tinycrypt [online]. 2017 [cit. 2020-11-13]. Dostupné z: <*https://github.com/intel/tinycrypt/blob/master/LICENSE*>*

[32] *HLAVÁČEK, Jan. Bezpečnostní testování zařízení s Bluetooth. Brno, 2017. Dostupné také z: <https://www.vutbr.cz/studenti/zav-prace/detail/101958>. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Petr Dzurenda, Ph.D.*

[33] *Bluetooth Security Mode / levels / encryption [online]. 2018 [cit. 2020-12-09]. Dostupné z: <https://support.honeywellaidc.com/s/article/Bluetooth-Security-Mode-levels-encryption>*

[34] *Wi-Fi Channels, Frequencies, Bands & Bandwidths [online]. [cit. 2020-11-27]. Dostupné z: <https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/channels-frequencies-bands-bandwidth.php>*

[35] *Wi-Fi HaLow: Low power, long range Wi-Fi® for IoT [online]. [cit. 2020-11-27]. Dostupné z: <https://www.wi-fi.org/discover-wi-fi/wi-fi-halow>*

[36] *Wireless Security Protocols: WEP, WPA, WPA2, and WPA3 [online]. [cit. 2021-11-27]. Dostupné z: <https://www.netspotapp.com/wifi-encryption-and-security.html>*

[37] *Research on unsecured Wi-Fi networks across the world [online]. 2016 [cit. 2020-11-27]. Dostupné z: <https://securelist.com/research-on-unsecured-wi-fi-networks-across-the-world/76733/>*

[38] *KÖNIG, Petr. Realizace laboratorní úlohy se systémem ZigBee. Brno, 2017. Dostupné také z: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=150891>. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky. Vedoucí práce Ing. Jiří Miloš, Ph.D.*

[39] *Maximizing Security in ZigBee Networks [online]. 2017 [cit. 2020-11-27]. Dostupné z: <https://www.nxp.com/docs/en/supporting-information/MAXSECZBNETART.pdf>*

[40] *RESLER, Tomáš. Návrh domácí brány pro zařízení IoT využívající technologii Z-Wave. Brno, 2018. Dostupné také z: <https://www.vutbr.cz/studenti/zav-prace/detail/118218?zp_id=118218>. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Doc. Ing. Jiří Hošek, Ph.D.*

[41] *Mandatory Security Implementation for Z-Wave IoT Devices Takes Effect [online]. 2017 [cit. 2020-11-27]. Dostupné z: <https://www.iotevolutionworld.com/smart-home/articles/430956-mandatory-security-implementation-z-wave-iot-devices-takes.htm>*

[42] *Z-Wave SDK 6.71: INTRODUCING S2 SECURITY [online]. [cit. 2020-11-27]. Dostupné z: <https://www.silabs.com/documents/public/presentations/PMP13827-2.pdf>*

[43] *OBRTÁČ, Tomáš. Návrh komplexního senzoru pro LoRa síť. Brno, 2016. Dostupné také z: <https://www.vutbr.cz/studenti/zav-prace/detail/100937?zp_id=100937>. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Ing. Jan Chalupa.*

[44] *LoRaWAN™ SECURITY [online]. 2019 [cit. 2020-11-27]. Dostupné z: <https://lora-alliance.org/sites/default/files/2019-05/lorawan_security_whitepaper.pdf>*

[45] *Sigfox Technical Overview: chapter 5 Security overview [online]. 2018 [cit. 2020-11-27]. Dostupné z: <https://www.avnet.com/wps/wcm/connect/onesite/03aebfe2-98f7-4c28-be5f-90638c898009/sigfox-technical-overview.pdf?MOD=AJPERES&CVID=magVa.N&CVID=magVa.N&CVID=magVa.N>*

[46] *VirtualBox. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-11-19]. Dostupné z: <https://cs.wikipedia.org/wiki/VirtualBox>*

[47] *Download VirtualBox [online]. [cit. 2020-11-19]. Dostupné z: <https://www.virtualbox.org/wiki/Downloads>*

[48] *Stáhnout Ubuntu pro osobní počítače [online]. [cit. 2020-11-19]. Dostupné z: <https://www.ubuntu.cz/ziskat-ubuntu/stahnout-desktop/>*

[49] *ESP8266 / ESP8285 [online]. [cit. 2020-11-19]. Dostupné z: <https://api.riot-os.org/group__cpu__esp8266.html>*

[50] *How to setup RIOT OS on Arduino with VirtualBox. Youtube [online]. [cit. 2020-12-03]. Dostupné z: <https://youtu.be/tS5W9COQOcI>*

[51] *How to setup RIOT OS on ESP8266 with VirtualBox. Youtube [online]. [cit. 2020-12-03]. Dostupné z: <https://youtu.be/DpDB9VVT4LU>*

[52] *ESP8266 does not respond [online]. [cit. 2020-11-22]. Dostupné z: <https://github.com/RIOT-OS/RIOT/issues/15137>*

[53] *Arduino Nano. Components101 [online]. 2018 [cit. 2020-11-08]. Dostupné z: <https://components101.com/microcontrollers/arduino-nano>*

[54] *NodeMCU ESP8266. Components101 [online]. 2020 [cit. 2020-11-08]. Dostupné z: <https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet>*

[55] *ESP32 - DevKitC. Components101 [online]. 2018 [cit. 2020-11-22]. Dostupné z: <https://components101.com/microcontrollers/esp32-devkitc>*

[56] *ESP32 vs ESP8266 – Pros and Cons. Maker Advisor [online]. 2020 [cit. 2020-11-22]. Dostupné z: <https://makeradvisor.com/esp32-vs-esp8266/>*

[57] *STM32F103C8T6 - Blue Pill Development Board. Components101 [online]. 2020 [cit. 2020-11-08]. Dostupné z: <https://components101.com/microcontrollers/stm32f103c8t8-blue-pill-development-board>*

[58] *Blackpill and Bluepill common [online]. [cit. 2020-11-22]. Dostupné z: <https://api.riot-os.org/group__boards__common__blxxpill.html>*

[59] *HAJNÝ, J.; DZURENDA, P.; CASANOVA MARQUÉS, R.; MALINA, L. Privacy ABCs: Now Ready for Your Wallets!. In Proceedings of The 19th International Conference on Pervasive Computing and Communications (IEEE PerCom 2021). 2021. s. 686-691. ISBN: 978-1-6654-0424-2.*

[60] *HAJNÝ, J.; DZURENDA, P.; CAMENISCH, J.; DRIJVERS, M. Fast Keyed-Verification Anonymous Credentials on Standard Smart Cards. In ICT Systems Security and Privacy Protection. Springer Nature Switzerland, 2019. s. 286-298. ISBN: 978-3-030-22312-0.*

[61] *CVRČEK, Tadeáš. Kryptografie na platformě Arduino. Brno, 2020. Dostupné také z: <https://www.vutbr.cz/studenti/zav-prace/detail/125926>. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Petr Dzurenda, Ph.D.*

[62] *MQTT: The Standard for IoT Messaging [online]. [cit. 2021-5-3]. Dostupné z: <https://mqtt.org/>*

[63] *ŠŤOVÍČEK, Petr. Universal security solution for IoT data collection systems. Proceedings I of the 27th Conference STUDENT EEICT 2021. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2021. ISBN 978-80-214-5943-4.*

[64] *MQTT Security Fundamentals [online]. [cit. 2021-5-4]. Dostupné z: <https://www.hivemq.com/mqtt-security-fundamentals/>*

[65] *BMP280/BME280 temperature, pressure and humidity sensor [online]. [cit. 2020-12-02]. Dostupné z: <https://riot-os.org/api/group__drivers__bmx280.html>*

[66] *Raspberry Pi 4 [online]. [cit. 2021-5-7]. Dostupné z: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>*

[67] *PAHO MQTT framework [online]. [cit. 2021-5-9]. Dostupné z: <http://api.riot-os.org/group__pkg__paho__mqtt.html>*

[68] *Micro-ECC for RIOT [online]. [cit. 2021-5-9]. Dostupné z: <http://doc.riot-os.org/group__pkg__micro__ecc.html>*

[69] *Crypto [online]. [cit. 2021-5-9]. Dostupné z: <https://doc.riot-os.org/group__sys__crypto.html>*

[70] *Eclipse Mosquitto [online]. [cit. 2021-5-9]. Dostupné z: <https://mosquitto.org/>*

[71] *PyCryptodome [online]. [cit. 2021-5-9]. Dostupné z: <https://pycryptodome.readthedocs.io/en/latest/>*

[72] *OpenHAB: empowering the smart home [online]. [cit. 2021-5-9]. Dostupné z: <https://www.openhab.org/>*

Seznam symbolů, veličin a zkratek

IoT Internet of Things

RFID Radio-frequency identification

DDoS Distributed denial-of-service

DES Data Encryption Standard

3DES Triple Data Encryption Algorithm, též TDES, TDEA, Triple DEA

NIST National Institute of Standards and Technology

AES Advanced Encryption Standard

RSA Rivest–Shamir–Adleman

TLS Transport Layer Security

PGP Pretty Good Privacy

VPN Virtual private network

DH Diffie–Hellman key exchange

DSA Digital Signature Algorithm

ECDH Elliptic-curve Diffie–Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

DNSSEC Domain Name System Security Extensions

CPU Central Processing Unit

RAM Random-Access Memory

GNU GNU's Not Unix!

NFC Near-Field Communication

PAN Personal Area Network

BLE Bluetooth Low Energy

PIN Personal Identification Number

WLAN Wireless local area network

SSID	Service Set IDentifier
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
TKIP	Temporal Key Integrity Protocol
LoRa	Long-Range
LPWAN	Low Power Wide Area Network
CMAC	Cipher-based Message Authentication Code
CTR	Counter
MAC	message authentication code
USB	Universal Serial Bus
PWM	Pulse-width modulation
TTL	Transistor-Transistor Logic
GPIO	General-purpose input/output
ADC	Analog-to-digital converter
PCB	Printed circuit board
ECB	Electronic Code Book
CCM	Counter with cipher block chaining message authentication code
KVAC	Keyed-Verification Anonymous Credentials
RKVAC	Revocable KVAC
MQTT	Message Queuing Telemetry Transport
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual local area network
KMS	Key management system
DLP	Discrete Logarithm Problem

ECDLP Elliptic curve discrete logarithm problem

CBC-MAC cipher block chaining message authentication code

AAD Additional authenticated data

XOR Exclusive or

GCM Galois/Counter Mode

ACL Access-control list

HTTPS Hypertext Transfer Protocol Secure

Seznam příloh

A Obsah přiloženého archivu

- `/` ..... Kořenový adresář přiloženého archivu
  - `benchmark` ..... Adresář se zdrojovými kódy výkonnostních testů
    - `AES`
      - `Crypto`
        - `AES128`
        - `AES-CCM-128`
      - `Relic`
        - `AES128`
      - `Tinycrypt`
        - `AES128`
        - `AES-CCM-128`
    - `EC`
      - `micro-ecc`
      - `Relic`
    - `ECDH`
      - `micro-ecc`
      - `Relic`
    - `ECDSA`
      - `micro-ecc`
      - `Relic`
    - `mod`
      - `micro-ecc`
      - `Relic`
    - `SHA256`
      - `Crypto`
      - `Relic`
      - `Tinycrypt`
    - `MAC`
    - `KVAC`
    - `RKVAC`
    - `AKA`
  - `mqtt-security`
    - `Broker` ..... Konfigurační soubor a soubor s ACL pravidly brokeru
    - `Klienti` ..... Zdrojové kódy jednotlivých klientů z implementovaného systému
    - `KMS` ..... Zdrojový kód KMS a konfigurační/databázový soubor KMS
    - `mqtt-security` ..... Zdrojový kód systému (pro klienty)
    - `keyGen.py` ..... Python skript generující klíče do systému
  - `benchmark.xlsx` ..... Výsledky výkonnostních testů
  - `RIOT_manual.pdf` ..... Návod na zprovozněný vývojového prostředí